CX-Supervisor
Script Language

Software Revision 1.2

Notice

OMRON products are manufactured for use according to proper procedures by a qualified operator and only for the purposes described in this manual.

The following conventions are used to indicate and classify precautions in this manual. Always heed the information provided in them. Failure to heed precautions can result in injury to people or damage to the product.

DANGER! Indicates information that, if not heeded, is likely to result in loss of life or serious injury.

WARNING Indicates information that, if not heeded, could possibly result in loss of life or serious injury.

Caution Indicates information that, if not heeded, could result in relatively serious or minor injury, damage to the product, or faulty operation.

OMRON Product References

All OMRON products are capitalised in this manual. The word "Unit" is also capitalised when it refers to an OMRON product, regardless of whether or not it appears in the proper name of the product.

The abbreviation "PLC" means Programmable Logic Controller and is not used as an abbreviation for anything else.

Visual Aids

The following headings appear in the left column of the manual to help you locate different types of information.

Note: Indicates information of particular interest for efficient and convenient operation of the product.

1, 2, 3… Indicates lists of one sort or another, such as procedures, checklists etc.

Represents a shortcut on the Toolbar to one of the options available on the menu of the same window.

Indicates a program must be started, usually by clicking the appropriate option under the standard Windows 'Start' button.

© OMRON, 2003

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, mechanical, electronic, photocopying, recording, or otherwise, without the prior written permission of OMRON.

All copyright and trademarks acknowledged.

No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.

About this Manual

This manual describes the script language syntax as a supplement to CX-Supervisor application user manual.

This manual contains the following:

- ***Chapter 1 Introduction***. An overview to this manual including special typographical conventions.
- ***Chapter 2 Expressions***. A description of the use of expressions within scripts.
- ***Chapter 3 Scripts***. An introduction to scripts and the type of scripts.
- ***Chapter 4 CX-Supervisor Script Language***. A detailed reference to the CX-Supervisor script language.
- ***Chapter 5 VBScript Language Reference***. A reference for the VBScript language.
- ***Chapter 6 Functions and Methods***. A detailed reference to the functions and methods available to script languages.
- ***Chapter 7 Script Examples***. A description of the script language in practice, using some examples.
- ***Chapter 8 Colour Palette***. A description of the colour palette that can be applied to certain script statements.
- ***Appendix A OPC Communications Control***. This appendix contains a list of the available component properties and gives details of the Visual Basic script interface.
- ***Appendix B Lite Communications Control***. This appendix contains a list of the available component properties and gives details of the Visual Basic script interface.
- ***Appendix C JScript Features***. This appendix provides a summary of the JScript features available for use with the ExecuteJScript and ExecuteJScriptFile script functions.
- ***Appendix D Obsolete Features***. This appendix provides a summary of the obsolete features, which remain enabled for backward compatibility.

A ***Glossary of Terms*** and ***Index*** are also provided.

TABLE OF CONTENTS

CX-Supervisor ..... Page

Chapter 1 – Introduction 1

Chapter 2 – Expressions 3

Chapter 3 – Scripts 7

Object 7

Page 7

Project 7

Chapter 4 – CX-Supervisor Script Language 9

Points 10

Logic and Arithmetic 12

Control Statements 15

Subroutines 22

Punctuation 23

Indirection within Script Commands and Expressions 26

Point Arrays within Script Commands and Expressions 27

Using Aliases 28

Chapter 5– VBScript Language reference 31

List of features 31

Chapter 6 – Functions and Methods 35

Object Commands 39

Page Commands 49

General Commands 50

Communications Commands 56

Point Commands 58

PLC Commands 67

Table of Contents Chapter 6 continued Page

Temperature Controller Commands 72
Alarm Commands 78
File Commands 83
Graph Commands 90
Recipe Commands 94
Report Commands 95
Text Commands 98
Event/Error Commands 104
Printer Commands 106
Security Commands 110
Data Logging Commands 112
Database Commands 118
Serial Port Functions 131
ActiveX Functions 134

Chapter 7 – Script Examples 138

Balloon Script 138

Chapter 8 – Colour Palette 142

Appendix A – OPC Communications Control 144

Component Properties 144
Script Interface 144
Functions 144

Appendix B – Lite Communications Control 146

Component Properties 146
Script Interface 146
Functions 146
PLC Memory Functions 150

Appendix C – JScript Features 152

Appendix D – Obsolete Features 154
Glossary of Terms 159
Index 167

CHAPTER 1
Introduction

This reference manual describes the script language syntax as a supplement to the CX-Supervisor User Manual. It provides detailed definition of the syntax of CX-Supervisor scripts that drive project, page, object actions and CX-Supervisor expressions as used by objects and scripts.

Typographic conventions used in the examples in this reference manual are as follows:

- Script commands and reserved words are shown in the preferred case, which may be either lower-, upper- or mixed-case.
- Points are shown in lower-case. Objects are shown in upper-case.

The following terms are used in this reference manual:

- Application. A set of files, containing an executable file, that carry out certain tasks. This reference manual refers to the Microsoft Excel and Microsoft Word for Windows applications.
- Constant. A point or object within a script that takes only one specific value.
- Executable. A file that contains programs or commands, and has an '*.EXE' extension.
- Nesting. To incorporate one or more IF THEN ELSE/ELSEIF ENDIF statements inside a structure of the same kind.
- Operands. Constants or point variables.
- Operators. Relational, arithmetic, and logical statements, for instance '+', '<=' or 'AND'.
- Or ('|'). The '|' symbol is used to represent 'or', where there are two or more forms of the same syntax.
- Point Types. Either Boolean, Integer, Real or Text.
- Point Variable. A point or object within a script that may take different values.
- Strings. Data in the form of text delimited by quotation marks (" "), which can be assigned to a point.
- The '{' and '}' braces. Must be inserted around the argument command or an error is reported. An error is reported if there are spaces between braces.
- 'TRUE' and 'FALSE'. Refer exclusively to Boolean states, where Boolean state 0 is 'FALSE' and Boolean state 1 is 'TRUE'.

CHAPTER 2
Expressions

This chapter describes the use of expressions within scripts.

Expressions consist of operators and operands:

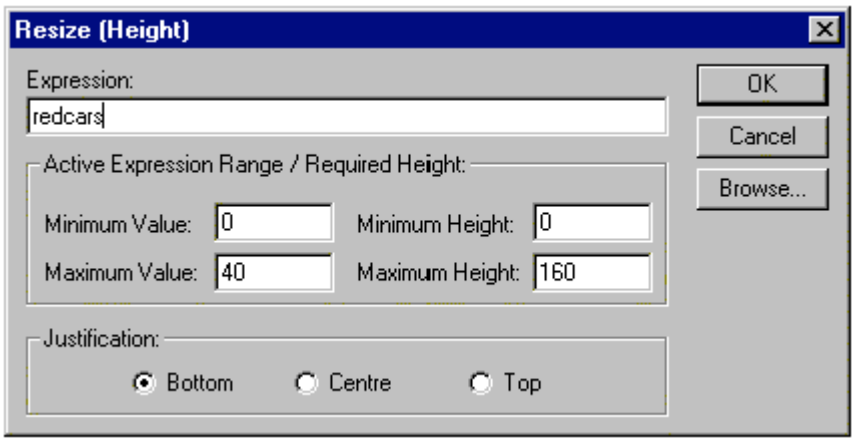
- Operators are relational, arithmetic, logical and include many functions.
- Operands are constants or point variables.

Expressions can be used in a script as part of a statement (refer to chapter 3 Scripts, chapter 4 CX-Supervisor Script Language, and Chapter 6 Functions and Methods). However expressions can be applied to the following actions directly using the associated *Expression:* or *Digital Expression:* field:

- Blink.
- Close page.
- Colour Change (Analogue).
- Colour Change (Digital).
- Display Status Text.
- Display Text Point.
- Display Value.
- Edit point value (Analogue).
- Edit point value (Digital).
- Edit point value (Text).
- Enable/Disable.
- Horizontal move.
- Horizontal percentage fill.
- Resize height.
- Resize width.
- Rotate.
- Show page.
- Vertical move.
- Vertical percentage fill.
- Visible.

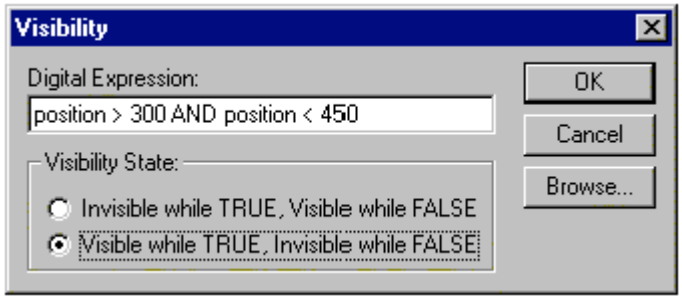
The following example of a simple expression contains a point ('redcars') attached to a particular object with an appropriate object action, Resize (Height). At runtime, once the value of the point has been met within the attributes declared within the *Active Expression Range/Required Height:* fields,

the current object is resized accordingly. This example is an Integer or Real example, whereby the value of the point either falls inside or outside the specified range. In this example, the point 'redcars' must fall between 0 and 40 for the expression to be met.

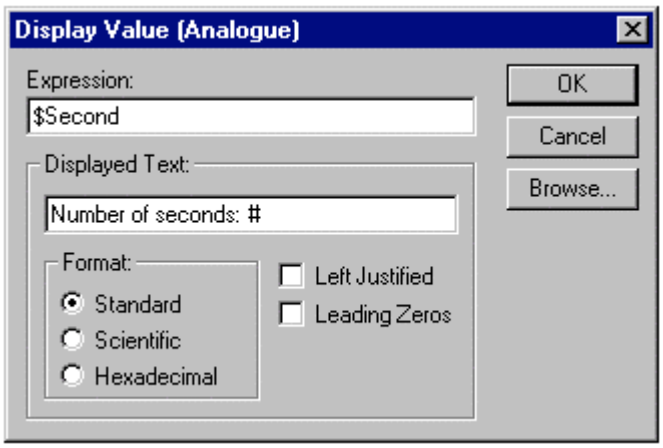


The following example of a more complex expression contains a test on point 'position'. If 'position' is more than 300 in value, and 'position' is less than 450 in value, i.e. the value of 'position' is between 300 and 450, then the expression has been met, and an action is initiated (in this instance the current object is made visible if the expression is met). This example is a Boolean example, whereby either the expression is met ('TRUE') or not met ('FALSE'). A Boolean value is always returned from a *Digital Expression:* field, as opposed to an *Expression:* field, which returns an Integer or Real value.

Operators used within this example are fully described in chapter 4, Logic and Arithmetic.



The following example of an expression contains a value point 'prompt' which is included at the value position denoted by a '#' symbol.



Refer to the *CX-Supervisor User Manual* for detailed dialog descriptions.

Note: Boolean Expressions execute when the expression is TRUE so it can be said that every Boolean expression has an inferred "== TRUE". Sometimes Boolean expressions can be difficult to read e.g. "bMyFlagPoint" or "BitMask & 0x80. It can help maintenance if this "== TRUE" is explicitly specified e.g. "bMyFlagPoint == TRUE" or "BitMask & 0x80 == TRUE".

Note: When using Boolean operators (e.g. ==, !=, &&, ||, |) never mix tests for Boolean and non Boolean operands. For example never use "bMyFlagPoint == 1" or "bMyFlagPoint == 0". Instead always test using the correct Boolean constant i.e. "TRUE" or "FALSE" for CX-Supervisor scripts, or "True" and "False" when using VBScript.

Note: On Condition scripts are only executed when the expression is TRUE. Sometimes this leads to peculiar results, for example using $Second as it will be executed when $Second changes to 59, and to 1 but *not* when it changes to 0. To execute a condition script any time a point changes, force the expression to always evaluate to TRUE for example "$Second || TRUE". This works because the $Second forces the expression to be tested when the point changes, but the || TRUE means the test will return TRUE regardless of the value of the point.

Note: Use array points in On Condition expressions with caution. The expression "MyArray[3] == 1" does not mean "execute every time the third element changes to 1". It means execute when *any* element of MyArray changes and the third element happens to be 1

Note: Using an array point without any index is the same as specifying element 0 i.e. MyArray actually means

Note: Use array points in On Condition expressions with caution. The expression "MyArray[3] == 1" does not mean "execute every time the third element changes to 1". It means execute when *any* element of MyArray changes and the third element happens to be 1

CHAPTER 3
Scripts

A CX-Supervisor script is a simple programming language used to manipulate points. Scripts can be created at different levels, at object level, page level or project level. Although the script code can be applied to all levels of script, there are subtle differences, described in the following paragraphs.

Object

If a script is executed as a runtime action of an object, then the script can affect the object of the action, or any other, depending on the actual content of the script.

Page

Page scripts are concerned with manipulating points and graphical objects that are used or included within that page. In other words page scripts are used to drive a number of actions on the occurrence of a particular event. These actions may manipulate several graphical objects on one page.

Project

Scripts can be applied to a project to manipulate points. These scripts are associated with events that occur throughout the whole operating session

CHAPTER 4
CX-Supervisor Script Language

This chapter describes the CX-Supervisor script language syntax. It provides a detailed definition of the syntax of CX-Supervisor scripts that drive project, page and object actions, and CX-Supervisor expressions as used by objects and scripts. In conjunction with the script functions and methods described in Chapter 6, the CX-Supervisor script language provides a very powerful, compiled, fast and full featured programming language.

The following table describes the script language syntax at a glance.

Function Name	Function Type	Type	Remarks
&, |, ^, <<, >>	bitwise operators	All	Applies bitwise expressions
(objects)	statement	OP	Specifies an object name for modification or test.
(points)	statement	All	Specifies a point name for modification or test.
+, -, *, /, %, =, ++, --	arithmetic operators	All	Applies arithmetic expressions.
<, >, <=, >=, ==, !=	relational operators	All	Applies relational expressions.
AND	logical operators	All	Applies logical expressions.
CALL	statement	All	Call a subroutine
DO LOOP WHILE UNTIL EXIT DO	statement	Scr	Script segment to be repeated
FALSE	Boolean state	Scr	Applies Boolean expression.
FOR TO STEP NEXT EXIT FOR	statement	Scr	Script segment to be repeated
IFTHEN ELSE\ELSEIF ENDIF	statement	Scr	Applies a test to a script.
OR	logical operators	All	Applies logical expressions.
NOT	logical operators	All	Applies logical expressions.
REM	statement	Scr	Remarks on line or lines of script.
RETURN	statement	Scr	Stops sequential execution of script.
SELECT CASE/END SELECT	statement	Scr	Applied to complex tests.
TRUE	Boolean state	Scr	Applies Boolean expression.

The ‘Type’ column refers to the types of script and expression the function can be applied to. ‘All’ refers to both expressions and scripts. ‘Scr’ refers to scripts only. ‘OP’ refers to Object and Page scripts only.

Points

Basic Point Assignment

Syntax

```
pointname = expression
```

Remarks

Argument	Description
pointname	The point name to be assigned a value.
expression	The value to be assigned to *pointname*. The expression may be of type Boolean, Integer, Real or Text.

Typical Examples

```
count = 100
```

The Integer or Real point ‘count’ is assigned the value 100.

```
result = TRUE
```

The Boolean point ‘result’ is assigned the state ‘‘TRUE’’.

```
name = "Valve position"
```

The Text point ‘name’ is assigned the associated text, contained within quotation marks.

References

Refer to chapter 4, Punctuation for details of the use of quotation marks.

Further Point Assignment

Syntax

```
pointname = expression
```

Remarks

Argument	Description
pointname	The point name to be assigned a value.
expression	The value to be assigned to *pointname*. The expression may be of type Boolean, Integer or Real and can include other points, logical or arithmetical expressions. Mathematical precedence is applied as follows: • Parenthesis (highest). • Unary minus and NOT logical operator. • Multiplication, division and modulus. • Addition and subtraction. • Greater than, less than, greater than or equal to, and less than or equal to relational operators. • Shift Left (SHL) and Shift Right (SHR). • Equal to and not equal to relational operators. • Bitwise AND, XOR, OR. • AND logical operator, OR logical operator (lowest).

Typical Examples

```
lift = height + rate/5.0
```

The Integer or Real point 'lift' is assigned the value calculated by the value of point 'rate' divided by 5, plus the value of point 'height'. Precedence can be changed by the introduction of parenthesis.

```
lift = lift - 0.2
```

The Integer or Real point 'lift' is assigned the value calculated by the current value of point 'lift' minus 0.2.

```
distance = distance * time
```

The Integer or Real point 'distance' is assigned the value calculated by the current value of point 'distance' multiplied by point 'time'.

References

Refer to chapter 4, Logic and Arithmetic for details of the use of arithmetic and logic functions. Refer to chapter 4, Punctuation for details of the use of parenthesis.

Logic and Arithmetic

Arithmetic Operators

Syntax

```
pointname = expression
```

Remarks

Argument	Description
pointname	The point name to be assigned a value based on an arithmetical expression.
expression	The value to be assigned to *pointname*. The expression may include the following operators with points and constants: • Addition '+'. • Subtraction '-'. • Multiplication '*'. • Division '/'. • Modulus '%'. • Increment '++'. • Decrement '--'.

Typical Examples

```
result = 60 + 20/5
```

The Integer or Real point 'result' is assigned the value calculated by the value of 20 divided by 5, plus 60.

```
lift = height + rate/5.0
```

The Integer or Real point 'lift' is assigned the value calculated by the value of point 'rate' divided by 5, plus the value of point 'height'. Precedence can be changed by the introduction of parenthesis.

References

Refer to chapter 4, Punctuation for details of the use of parenthesis.

Bitwise Operators

Syntax

```
pointname = expression
```

or

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
pointname	The pointname to be assigned a value based on the bitwise operation.
expression	The value to be assigned to *pointname*, or to be evaluated as a Boolean expression. The expression can include the following operators with points and constants: • Bitwise AND, 'BITAND' or '&'. • Bitwise OR, 'BITOR' or '|'. • Bitwise XOR, 'XOR' or '^'. • Bitwise Shift Left, 'SHL' or '<<'. • Bitwise Shift Right, 'SHR' or '>>'.

Typical Examples

```
MSB = value & 128
```

The Boolean point 'MSB' is set 'TRUE' if the binary representation of 'value' has the bit set which is worth 128.

```
Pattern = value << 2
```

The binary representation of 'value' is shifted left twice, and stored in 'pattern'. Each Shift Left operation has the effect of doubling the value, so two shifts quadruple the value.

Logical Operators

Syntax

```
pointname = expression
```

or

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
Pointname	The point name to be assigned a value based on a logical expression.
Expression	The Boolean value to be assigned to *pointname* or the Boolean value forming a conditional statement. The expression includes the following operators with points and constants: • And 'AND'. • Or 'OR'. • Not 'NOT'.

Typical Examples

```
flag = temp AND speed
```

The Boolean point 'flag' is assigned a value based on the logic of point 'temp' AND point 'speed'. If 'temp' and 'speed' are both not zero, 'flag' is set to 1, or ''TRUE''. A value of zero in either 'temp' or 'speed' supplies 'FALSE' or 0 to 'flag'.

```
IF flag AND temp AND speed THEN
  flag = FALSE
ENDIF
```

The Boolean point 'flag' is assigned 'FALSE', on the condition that 'flag' AND point 'temp' AND point 'speed' are all not zero. If the condition fails, then 'flag' is not assigned 'FALSE'.

References

Refer to chapter 4, Control Statements for details of the use of the IF THEN ELSE/ELSEIF ENDIF statements.

Relational Operators

Syntax

```
IF expression
```

or

```
DO WHILE expression
```

or

```
DO UNTIL expression
```

Remarks

Argument	Description
Expression	The value forming a conditional statement. The expression may include the following operators with points and constants: • Greater than '>'. • Less than '<'. • Greater than or equal to '>='. • Less than or equal to '<='. • Not equal to '!='. • Equal to '=='.

Typical Example

```
IF fuel < 0 THEN
  fuel = 0
ENDIF
```

The point 'fuel' is assigned the value 0 on the condition that currently, 'fuel' is less than 0. If 'fuel' is not less than 0, then it is not assigned the new value.

References

Refer to chapter 4, Control Statements for details of the use of the IF THEN ELSE/ELSEIF ENDIF statements.

Control Statements

Simple Conditional Statements

Syntax

```
IF condition THEN
  statementblock1
ENDIF
```

or

```
IF condition THEN
  statementblock1
ELSE
  statementblock2
ENDIF
```

Remarks

Argument	Description
Condition	The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string.
Statementblock1	One or more statements which are performed if the *condition* is met.
Statementblock2	One or more statements which are performed if the *condition* is not met.

Typical Examples

```
IF fuel < 0 THEN
  fuel = 0
ENDIF
```

Provided Integer point 'fuel' is less than 0, then it is assigned the value 0.

```
IF burner THEN
  fuel = fuel - rate
ENDIF
```

Provided Boolean point 'burner' is ''TRUE'', then Integer point 'fuel' is assigned a new value. It is also possible to apply 'IF burner == TRUE THEN' as the first line, with identical results.

```
IF distance > 630 AND distance < 660 AND lift >= -3 THEN
  winner = TRUE
  burner = FALSE
ENDIF
```

Provided that Integer point 'distance' is greater in value than 630 AND 'distance' is less in value than 660 (i.e. 'distance' is a value between 630 and 660) AND point 'lift' is greater than or equal to -3, then Boolean points 'winner' and 'burner' are assigned new values.

```
IF burner AND fuel > 0 AND rate > 0 THEN
  fuel = fuel - rate
ELSE
  lift = 0
  altitude = 0
ENDIF
```

Provided that Boolean point 'burner' is ''TRUE'' AND points 'fuel' and 'rate' are greater in value than 0, then 'fuel' is assigned a new value. Otherwise points 'lift' and 'altitude' are assigned a new value.

References

Refer to chapter 4, Punctuation, Indentation for details on the layout of code.

Nested Conditional Statements

Syntax

```
IF conditionA THEN
  statementblock1
  IF conditionB THEN
    statementblock3
  ENDIF
ELSE
  statementblock2
ENDIF
```

or

```
IF conditionA THEN
  statementblock1
  IF conditionB THEN
    statementblock3
  ELSE
    statementblock4
  ENDIF
ELSE
  statementblock2
ENDIF
```

or

```
IF conditionA THEN
  statementblock1
ELSEIF conditionB THEN
  statementblock3
ENDIF
```

or

```
IF conditionA THEN
  statementblock1
ELSE
  statementblock2
  IF conditionB THEN
    statementblock3
  ELSE
    statementblock4
  ENDIF
ENDIF
```

Remarks

Argument	Description
conditionA	The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string.
conditionB	This **condition** is nested in the first condition, either on a successful or unsuccessful evaluation of *conditionA*. The condition is made up of points and constants, using relational, logical or arithmetical notation as a test. The condition can evaluate Boolean state 'TRUE' and 'FALSE', Integer or Real numbers, or a text string. There is no limit to the number of nested conditional statements.
statementblock1	One or more statements which are performed if *conditionA* is met.
statementblock2	One or more statements which are performed if *conditionA* is not met.
statementblock3	One or more statements which are performed if *conditionB* is met.
statementblock4	One or more statements which are performed if *conditionB* is not met.

Typical Examples

```
IF burner AND fuel > 0 AND rate > 0 THEN
  lift = lift + rate/5
ELSE
  count = 1
  IF altitude > 140 THEN
    lift = lift - 0.2
  ENDIF
ENDIF
```

Provided a successful evaluation has been made to points 'burner' AND 'fuel' AND 'rate', point 'lift' is updated with the current value of rate divided by 5 plus 'lift'. Otherwise, a further evaluation is required on point 'altitude'. If 'altitude' is currently greater than 140, then 'lift' is decremented by 0.2.

```
IF burner AND fuel > 0 AND rate > 0 THEN
  lift = lift + rate/5
ELSE
  IF altitude > 140 THEN
    lift = lift - 0.2
  ENDIF
ENDIF
```

```
IF burner AND fuel > 0 AND rate > 0 THEN
  lift = lift + rate/5
ELSEIF altitude > 140 THEN
  lift = lift - 0.2
ENDIF
```

These two examples are identical. The use of the ELSEIF statement combines the ELSE statement and the IF/ENDIF statements for brevity. It is acceptable to have more than one ELSEIF statement in an IF THEN ELSE/ELSEIF ENDIF construct.

References

Refer to chapter 4, Punctuation for details of the use of indentation.

Case Select

Syntax

```
SELECT CASE expression
  CASE expression
    statementblock1
  CASE expression
    statementblock2
  CASE expression
    statementblock3
END SELECT
```

or

```
SELECT CASE expression
  CASE expression
    statementblock1
  CASE expression
    statementblock2
  CASE ELSE
    statementblock3
END SELECT
```

Remarks

Argument	Description
expression	The *expression* may be a point, or a calculation of constants and/or points that produces a result.
statementblock1	One or more statements that are only performed if the preceding CASE expression is met.
statementblock2	One or more statements that are only performed if the preceding CASE expression is met.
statementblock3	One or more statements that are only performed if the preceding CASE expression is met.

Typical Examples

```
SELECT CASE colourvalue
  CASE 1
    colour (blue)
  CASE 2
    colour (green)
  CASE 3
    colour (cyan)
  CASE ELSE
    colour (0)
END SELECT
```

This example shows the assignment of a colour according to the value of a point. The value of Integer point 'colourvalue' is evaluated and compared with each case until a match is found. When a match is found, the sequence of actions associated with the CASE statement is performed. When 'colourvalue' is 1, the colour given to the current object is blue, when 'colourvalue' is 2, the colour given to the current object is green, when 'colourvalue' is 3, the colour given to the current object is cyan. If 'colourvalue' falls outside the integer range 1—3, then the colour given is 0 (black). Like ELSE and ELSEIF, the CASE ELSE statement is optional.

```
SELECT CASE TRUE
  CASE temperature > 0 AND temperature <= 10
    colour (blue)
  CASE temperature > 10 AND temperature <= 20
    colour (green)
  CASE temperature > 20 AND temperature <= 30
    colour (red)
  CASE ELSE
    colour (white)
ENDSELECT
```

In this example, instead of using a point as the condition as with the previous example, the value is the condition — in this case Boolean state ''TRUE'' — with the integer point 'temperature' being tested at each case. If it is ''TRUE'' that 'temperature' is between 0 and 10, then the current object is set to blue, or if it is ''TRUE'' that 'temperature' is between 11 and 20, then the current object is set to green, or if it is ''TRUE'' that 'temperature' is between 21 and 30, then the current object is set to red. If none of these CASE statements are met, then the current object is set to white. Like ELSE and ELSEIF, the CASE ELSE statement is optional.

References

Refer to chapter 6, Object Commands for details of applying attributes to an object and for the use of the Colour object command. Refer to chapter 8, Colour Palette for details of the Colour Palette colour designation.

FOR... NEXT Loop

Syntax

```
FOR pointname = startpt TO endpt STEP steppt
  statementblock1
NEXT
```

Remarks

Argument	Description
pointname	The pointname to be used as the loop counter.
startpt	The initial setting of *pointname*, and the first value to be used through the loop.
endpt	The last value to be used. The loop ends when *pointname* exceeds this value.
steppt	Amount to increase *pointname* by every pass of the loop. *Steppt* can be negative to count backwards providing *startpt* is larger than *endpt*. The STEP keyword and variable may be omitted in which case *pointname* is incremented at each pass of the loop (identical to adding STEP 1).

Typical Examples

```
FOR loopcount = 0 TO 100
  Ellipse_1.vertical%fill = loopcount
NEXT
```

In this example, 'Ellipse_1' is gradually filled 100 times.

```
FOR loopcount = 100 TO 0 STEP -5
  Ellipse_1.vertical%fill = loopcount
NEXT
```

In this example, the fill for 'Ellipse_1' is gradually removed 20 times (100 times/-5).

Note: Loop statements should be used with caution, as they consume processor time while they are running and some other parts of the system may not be updated.

DO WHILE/UNTIL Loop

Syntax

```
DO WHILE expression
  statementblock
LOOP
```

or

```
DO
  statementblock
LOOP WHILE expression
```

or

```
DO UNTIL expression
  statementblock
LOOP
```

or

```
DO
  statementblock
LOOP UNTIL expression
```

Remarks

Argument	Description
expression	The *expression* may be a point, or a calculation of constants and/or points that produces a result.
statementblock	One or more statements to be executed multiple times depending on expression.

Typical Example

```
DO WHILE dooropen == TRUE
  Message ("You must shut the door before continuing")
LOOP
DO
  nextchar = Mid (Mystring, position, 1)
  position = position + 1
LOOP UNTIL nextchar = "A"
```

Note: Loop statements should be used with caution, as they consume processor time while they are running and some other parts of the system may not be updated.

Subroutines

Call

Syntax

```
CALL subroutine (arguments)
```

Remarks

Argument	Description
subroutine	The name of the subroutine defined at project level.
arguments	The list of arguments required by the *subroutine* separated by commas. Each argument may be a pointname, constant, arithmetical or logical expression or any valid combination.

Typical Example

```
CALL MySub ($Second, "Default", 2 + Int1)
```

Return

Syntax

```
RETURN
```

Typical Example

```
IF limit > 1000 THEN
  RETURN
ELSE
  value = limit
ENDIF

REM final part of script
POLYGON_1.COLOUR = red
ELLIPSE_5.WIDTH = value
```

The integer point 'limit' is tested for its value. If its value exceeds 1000, then the condition is met, and the RETURN command is executed. All statements after the RETURN command are ignored. If the value of integer point 'limit' does not exceed 1000, then the RETURN command is not executed, and statements after the RETURN command are performed.

References

Refer to the *CX-Supervisor User Manual* for the use of the RETURN statement for Recipe validation.

Punctuation

Command String Delimiters

Description

Alternative string delimiters allowing string to contain quote " characters.

Syntax

```
{Some "string" text}
```

Typical Example

```
Message({Error: "Invalid Function" occurred})
```

The '{' and '}' braces inserted around the whole strings allows the actual text in the string to contain quotes which will be displayed normally. They can be used in any situation where quotes can be used whether or not embedded quotes are required. However, for clarity the quote characters should be used by preference.

Indentation

Typical Examples

```
IF burner AND fuel > 0 AND rate > 0 THEN
lift = lift + rate/5
ELSE
IF altitude > 140 THEN
lift = lift - 0.2
ENDIF
ENDIF
```

```
IF burner AND fuel > 0 AND rate > 0 THEN
    lift = lift + rate/5
ELSE
    IF altitude > 140 THEN
        lift = lift - 0.2
    ENDIF
ENDIF
```

Both examples provide identical functionality, but the use of indentation, either spaces or tabs to show the construction of the statements aids readability.

The use of the ELSEIF statement in this example was omitted for clarity.

Multiple Commands

Typical Examples

```
count = 75
result = log(count)
```

```
count = 75 : result = log(count)
```

Both examples provide identical functionality, but the use of the colon between statements allows both to reside on the same line.

Parenthesis

Typical Examples

```
result = 20 + 30 * 40
```

The result is 1220.

```
result = (20 + 30) * 40
```

The values in parenthesis are calculated first. The result is 2000.

References

Refer to chapter 4, Logic and Arithmetic, Arithmetric Operations for further details.

Quotation Marks

Typical Examples

```
name = "Valve position"
```

The Text point 'name' is assigned associated text, contained within quotation marks. Quotation marks must be used in this instance.

```
Message("This text to be displayed as a message.")
```

Passing static text as arguments to functions.

```
BlueCarsAck = IsAlarmAcknowledged("BLUEPAINT")
```

The point 'BlueCarsAck' is assigned a Boolean state based on the alarm 'BLUEPAINT'. Quotation marks must be used for an alarm name.

Remarks

Syntax

```
REM | rem comment
```

or

```
'comment
```

Remarks

Argument	Type	Description
Comment	---	Descriptive text.

Typical Examples

```
REM The following statement adds two numbers
result = 45 + 754
```

```
result = 45 + 754   'add two numbers
```

Indirection within Script Commands and Expressions

It is possible to use text points directly or indirectly in place of literal string arguments within scripts and expressions. For instance, each of the following commands has the same effect:

- Using a string literal;

```
PlayOLE("ole_1", 0)
```

- Using a textpoint directly;

```
textpoint = "ole_1"
PlayOLE(textpoint, 0)
```

- Using a textpoint indirectly via the '^' notation.

```
text = "ole_1"
textpoint = "text"
PlayOLE(^textpoint, 0)
```

It is possible to use text points indirectly in place of point name arguments within script commands. For instance, each of the following commands has the same effect:

- Using a point name directly;

```
verbnumber = 0
PlayOLE("ole_1", verbnumber)
```

- Using a textpoint indirectly via the '^' notation.

```
verbnumber = 0
textpoint = "verbnumber"
PlayOLE("ole_1", ^textpoint)
```

An example using Indirection

The value of point indirection can be seen in a situation where it is necessary to dynamically change the pointname that an object is linked to. In the following example a toggle button is configured to control the Boolean state of one of four points:

- The four Boolean points to be controlled are called 'motor1', 'motor2', 'motor3' and 'motor4'.
- The text point 'textpoint' is used to store the name of the Boolean point to be controlled.
- The text point 'text' is used to store the string value of the integer point 'index'
- The integer point 'index' (which has a range 1-4) is used to dynamically change the point being controlled.
- Access to any of the four Boolean points 'motor1', 'motor2', 'motor3', 'motor4' can be achieved by applying indirection to 'textpoint' using the '^' notation and changing the contents of 'textpoint'.

For instance, in order to dynamically change the Boolean point a toggle button is linked to follow these steps.

1, 2, 3…

1. Link the toggle button to a textpoint using indirection e.g. ^textpoint.
2. Link the following script code to run as required. e.g. on clicking a button.
   - Text = ValueToText(index)
   - TextPoint = "motor" + text
3. The ValueToText function converts the integer value of the point 'index' into a string held in the textpoint 'text'. Therefore the point 'text' contains either '1', '2', '3' or '4'. The expression 'motor' + text appends the contents of the point 'text' to the literal string 'motor'. Therefore 'textpoint' contains either 'motor1', 'motor2', 'motor3' or 'motor4' dependant on the value of 'index'. Change the value of the 'index' to determine which Boolean point to control. e.g. via the Edit Point Value (Analogue) animation.

Point Arrays within Script Commands and Expressions

It is possible to access the elements of a point array directly or indirectly from within scripts or expressions.

- Setting the value of an array point directly;

```
arraypoint[2] = 30
```

- Getting the value of an array point directly;

```
value = arraypoint[2]
```

- Setting the value of an array point using indirection;

```
textpoint = "arraypoint"
^textpoint[2] = 30
```

- Getting the value of an array point using indirection;

```
textpoint = "arraypoint"
value = ^textpoint[2]
```

An example using Point Arrays

The value of array points can be seen in a situation where it is necessary to dynamically change the pointname that an object is linked to. In the following example a toggle button is configured to control the Boolean state of one of four elements of an array point.

The Boolean array point 'motor' is configured to contain 4 elements.

The integer point 'index' (which has a range 0-3) is used to dynamically change the element of the point being controlled.

In order to dynamically change the element of a Boolean point that a toggle button is linked to follow these steps.

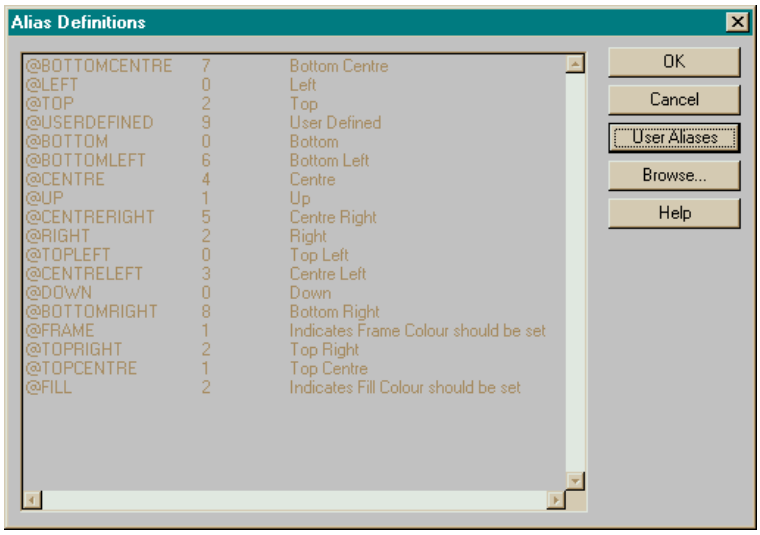
1, 2, 3…

1. Link the toggle button to an array point. e.g. 'motor[index]'.
2. Change the value of the 'index' to determine which element of the Boolean point to control. e.g. via the Edit Point Value (Analogue) animation.

Using Aliases

This facility is used to declare an alias - that is, to define a text string that can be used in place of another text string or a number within any script or expression. The Alias Definitions dialog is displayed by selecting the "Alias Definition..." option from the Project menu. It can also be displayed if "Aliases..." is selected from the script editor. The dialog displays either the User defined aliases or the preset System aliases and is toggled between these two displays by pressing the User/System Alias button.

The following illustration shows the Alias Definitions dialog displaying a number of User defined aliases. The System aliases are pre-defined and can not be edited or added to.



Syntax:

```
@AliasName    Alias definition    'optional comment
```

Remarks:

Argument	Type	Description
@AliasName	string	The string name of the alias
Alias definition	string	This is a string representing the actual text or expression of the expanded alias.
' comment	string	This is an optional comment.

The @ symbol at the beginning of each line initiates each alias command. For example, the text string *@SomePoint* could be used to represent any sequence of characters in a script or expression – e.g. it could be defined as:

```
@SomePoint = InArray[1]
```

or even

```
@SomePoint = Inarray[1] + Inarray[2] / 2
```

This is an easy way of identifying the individual members of array points. It can also be used to associate names with numbers, for example,

```
@SecondsPerDay = 86400
```

Alias definitions are stored in a simple text file in the project directory, called <project name>.pre. The format of the file consists of any number of lines such as:

```
@Test1 = InArray[12] * 10
```

i.e. an @ symbol followed by the name of the alias, then an equals sign (or space), followed by the definition of the alias. Anything that follows the last apostrophe (') symbol on a line is interpreted as a comment. Any line which does not start with the @ symbol is also assumed to be a comment.

Typical Examples

```
Declare boiler temperatures
@BoilerTemp1 = InArray[0] ' for boiler room 1
@BoilerTemp2 = InArray[1] ' for boiler room 2
@SecondsPerMinute  = 60   ' sets duration
```

Aliases may also be used to create a complicated expression such as

```
@HYPOTENUSE   sqrt(Opposite * Opposite + Adjacent * Adjacent)
'Calculates length of Hypotenuse
```

This can be used in a script in the following way:

```
Opposite = 8.45
Adjacent = 9.756
length = @HYPOTENUSE
```

where Opposite, Adjacent and length are all REAL points.

Note: Changing an alias definition after it has been used in an expression or script will not automatically change the result in the script. The appropriate script or expression where that alias is used must be accessed and recompiled by pressing the OK button in order to apply the changes.

CHAPTER 5
VBScript Language Reference

This chapter is a reference for the syntax of Microsoft Visual Basic scripting language called VBScript. These features are provided by the Windows Scripting Host, included by default with Windows 98, Windows ME, Windows 2000 and Windows XP and installed by Internet Explorer 4.0 and later. For Windows 95 and Windows NT, the Windows Scripting Host is available as a free download from Microsoft's Web site.

For a full User Guide, Language reference and details of the latest versions and support contact Microsoft at http://msdn.microsoft.com/scripting

List of Features:

Category	Keyword / Feature
Array handling	Array Dim, Private, Public, ReDim IsArray Erase LBound, UBound
Assignments	Set
Comments	Comments using ' or Rem
Constants/Literals	Empty Nothing Null True, False
Control flow	Do…Loop For…Next For Each…Next If…Then…Else Select Case While…Wend With
Conversions	Abs Asc, AscB, AscW Chr, ChrB, ChrW CBool, CByte CCur, Cdate CDbl, CInt CLng, CSng, CStr DataSerial, DateValue Hex, Oct Fix, Int Sgn TimeSerial, TimeValue
Date / Times	Date, Time DateAdd, DateDiff, DatePart DateSerial, DateValue Day, Month, MonthName

Category	Keyword / Feature
	Weekday, weekdayName, Year Hour, Minute, Second Now TimeSerial, TimeValue
Declarations	Class Const Dim, Private, Public, ReDim Function, Sub Property Get, Property Let, Property Set
Error Handing	On Error Err
Expressions	Eval Excute RegExp Replace Test
Formatting Strings	FormatCurrency FormatDateTime FormatNumber FormatPercent
Input / Output	InputBox LoadPicture MsgBox
Literals	Empty False Nothing Null True
Math	Atn, Cos, Sin, Tan Exp, Log, Sqr Randomize, Rnd
Miscellaneous	Eval Function Execute Statement RGB Function
Objects	CreateObject Err Object GetObject RegExp
Operators	Addition (+), Subtraction (-) Exporentiation (^) Modulus arithmetic (Mod) Multiplication (*), Division (/) Integer Division (\) Negation (-) String concatentation (&) Equality (=), Inequality (<>) Less Than (<), LessThan or Equal(<+) Greater Than (>) Greater Than or Equal To (>=) Is And, Or, Xor

Category	Keyword / Feature
	Eqv, Imp
Options	Option Explicit
Procedures	Call Function, Sub Property Get, Property Let, Property Set
Rounding	Abs Int, Fix, Round Sgn
Script Engine ID	ScriptEngine ScriptEngineBuildVersion ScriptEngineMajorVersion ScriptEngineMinorVersion
Strings	Asc, AscB, AscW Chr, ChrB, ChrW Filter, InStr, InStrB InStrRev Join Len, LenB LCase, UCase Left, LeftB Mid, MidB Right, RightB Replace Space Split StrComp String StrReverse LTrim, RTrim, Trim
Variants	IsArray IsDate IsEmpty IsNull IsNumeric IsObject TypeName VarType

CHAPTER 6
Functions and Methods

This chapter describes the Functions and Methods available to the scripting language. In most cases, this can be CX-Supervisor script, VBScript or JScript.

The following table describes the Functions and Methods at a glance.

Function Name	Function Type	Type	Remarks
AcknowledgeAlarm	alarm command	Scr	Acknowledges an alarm.
AcknowledgeAllAlarms	alarm command	Scr	Acknowledges all alarms.
AcknowledgeLatestAlarm	alarm command	Scr	Acknowledge the latest alarm.
Acos	unary function	All	Applies unary expression.
Asin	unary function	All	Applies unary expression.
Atan	unary function	All	Applies unary expression.
CancelForce	point command	Scr	Removes the forcing of values on a point.
Chr	text command	All	Displays a character based on the ASCII character set.
ClearAlarmHistory	alarm command	All	Clears the alarm history.
ClearErrorLog	event/error commands	All	Clears the error log.
ClearGraph	graph command	Scr	Clears the data from a trend or scatter graph.
ClearLogFile	Data Logging command	Scr	Clears a data log file
ClearSpoolQueue	printer command	All	Discards any queued messages or alarms.
close	object command	Scr	Closes a specified page.
CloseAlarmHistory	alarm command	All	Closes the current alarm history.
CloseAlarmStatus	alarm command	Scr	Closes the current alarm status.
CloseComponent	comms command	All	Closes a component for a PLC (e.g. CX-Server components).
CloseErrorLog	error command	Scr	Closes the currently open Error Log.
CloseFile	file command	Scr	Closes the open file.
CloseLogFile	Data Logging command	Scr	Closes a data log file
CloseLogView	Data Logging command	Scr	Closes the log viewer
ClosePLC	PLC command	Scr	Close communications with a PLC.
colour	object command	OP	Specifies a colour to an object.
CopyArray	point command	All	Copies the content of an array.
CopyFile	file command	Scr	Copies a specified file.
cos	unary function	All	Applies unary expression.
DeleteFile	file command	Scr	Deletes the specified file.
disable	object command	OP	Disables an object.
DisableGroup	point command	All	Prevents a group of points to be read or written.

Function Name	Function Type	Type	Remarks
DisablePoint	point command	Scr	Disables communications to a point.
display	object command	Scr	Displays a specified page.
DisplayAlarmHistory	alarm command	Scr	Displays the current alarm history.
DisplayAlarmStatus	alarm command	Scr	Displays the alarm status of all current alarms.
DisplayErrorLog	event command	Scr	Displays the current Error Log.
DisplayRecipes	recipe command	Scr	View the current recipes in the project.
DownloadPLCProgram	PLC command	All	Downloads specified files to the PLC.
DownloadRecipe	recipe command	Scr	Downloads a specified recipe.
EditFile	file command	All	Edits a specified file.
EditGraph	graph command	Scr	Displays a dialog used for editing trend graphs.
EnableAlarms	alarm command	All	Enables alarm functions.
EnableErrorLogging	error command	Scr	All actions become subject to Error Logging.
EnableGroup	point command	All	Permits a group of points to be read or written.
EnableOLE	comms command	Scr	Allows use of OLE functions.
EnablePLC	comms command	Scr	Allows use of PLC functions.
EnablePoint	point command	Scr	Enables communications to a point.
EnablePrinting	printer command	All	Permits printing of Alarms or messages.
ExportAndViewLog	Data Logging command	Scr	Exports data log and views
ExportLog	Data Logging command	Scr	Exports data log
FileExists	file command	All	Specifies the existence of a file.
Force	point command	Scr	Locks the value of a point.
ForceReset	point command	Scr	Sets a point value to 0.
ForceSet	point command	Scr	Sets a point value to 1.
FormatText	text command	All	Inserts text with standard ‘C’ formatting characters.
GenerateReport	report command	All	Produces a report based on a report template.
GetBit	point command	All	Retrieves a bit from a point.
GetPLCMode	PLC command	All	Retrieves the mode of a PLC.
GetPointValue	point command	Scr	Retrieves the value of a point.
GetSpoolCount	printer command	All	Specifies the number of alarms or messages queued for printing.
GetTextLength	text command	All	Specifies the number of characters in a text point.
height	object command	OP	Specifies the height of an object.
horizontal%fill	object command	OP	Specifies the horizontal fill of an object.
InputPoint	point command	Scr	Reads a value from a point.
IsAlarmAcknowledged	alarm command	Scr	Tests if a specified alarm has been acknowledged.

Function Name	Function Type	Type	Remarks
IsAlarmActive	alarm command	Scr	Tests if a specified alarm is currently active.
Left	statement	Scr	Extracts characters from the left of a string
log	unary function	All	Calculates the natural logarithm on a number.
log10	unary function	All	Calculates the base-10 logarithm on a number.
LogError	error command	Scr	Logs an error message with the error logger.
LogEvent	error command	Scr	Logs an event message with the error logger.
Login	security command	Scr	Logs a user into a run-time application.
Logout	security command	Scr	Logs a user out of a run-time application.
Message	text command	Scr	Outputs a string in a message box.
Mid	text command	Scr	Extracts a substring from a string.
move	object command	OP	Moves an object.
MoveFile	file command	Scr	Moves the specified file.
OpenComponent	comms command	All	Opens a component for a PLC (e.g. CX-Server components).
OpenFile	file command	Scr	Opens the specified file.
OpenLogFile	Data Logging command	Scr	Opens a data log file
OpenLogView	Data Logging command	Scr	Opens the Data Log Viewer
OpenPLC	PLC command	Scr	Opens communications with a PLC.
OutputPoint	point command	Scr	Displays the current value of a point.
PlayOLE	gen. command	Scr	Plays an OLE object.
PlaySound	gen. command	Scr	Plays a sound file.
PLCCommsFailed	PLC command	All	Specifies if the PLC communications have failed.
PLCMonitor	PLC command	Scr	Monitors a PLC.
PointExists	point command	All	Specifies the existence of a point.
PrintActivePage	gen. command	Scr	Prints the currently active page.
PrintFile	file command	Scr	Prints the specified file.
PrintMessage	text command	All	Prints messages to the configured 'Alarm/message printer'.
PrintPage	gen. command	Scr	Prints the specified page.
PrintReport	report command	All	Prints a report
PrintScreen	gen. command	Scr	Prints the current display screen.
PrintSpoolQueue	printer command	All	Prints all queued alarms or messages.
Rand	gen. command	Scr	Calculates a random number.
Read	file command	Scr	Reads data from an open file into a point.

Function Name	Function Type	Type	Remarks
ReadMessage	file command	All	Reads text from an external file.
Right	text command	Scr	Extracts characters from the right of a string.
rotate	object command	OP	Rotates an object.
RunApplication	gen. command	Scr	Runs the specified application.
RunHelp	gen. command	Scr	Runs the specified help file.
SaveGraph	graph command	All	Saves graph data to disk.
SelectFile	file command	All	Specifies a file name and path.
SetBit	point command	All	Sets a specific bit from a point.
SetPLCMode	PLC command	All	Sets the mode of a PLC.
SetPLCPhoneNumber	PLC command	All	Sets a phone number to a PLC.
SetPointValue	point command	Scr	Sets the value of a point.
SetPrinterConfig	printer command	All	Defines printer configuration.
SetupUsers	security command	Scr	Defines users and passwords for Login.
ShutDown	gen. command	Scr	Terminates CX-Supervisor.
sin	unary function	All	Applies unary expression.
Snapshot	graph command	Scr	Stores the current state of a trend graph.
sqrt	unary function	All	Applies unary expression.
StartGraph	graph command	Scr	Starts the recording of a graphs data.
StartLogging	Data Logging command	Scr	Starts a data set logging.
StopGraph	graph command	Scr	Stops the recording of a graphs data.
StopLogging	Data Logging command	Scr	Stops a data set logging.
tan	unary function	All	Applies unary expression.
TCAutoTune	temp. controller command	All	Starts or stops a temperature controller auto-tune operation.
TCBackupMode	temp. controller command	All	Defines how a temperature controller stores internal variables.
TCGetStatusParameter	temp. controller command	All	Retrieves the temperature controller status parameter.
TCRemoteLocal	temp. controller command	All	Defines the operational mode of a temperature controller.
TCRequestStatus	temp. controller command	All	Retrieves the temperature controller status.
TCReset	temp. controller command	All	Resets the temperature controller.
TCRspLsp	temp. controller command	All	Defines the setpoint mode used by the temperature controller.
TCRunStop	temp. controller command	All	Defines either auto-output mode shift or manual output mode shift.
TCSaveData	temp. controller command	All	Saves data associated with the temperature controller.
TCSettingLevel1	temp. controller command	All	Performs a settinglevel function for the temperature controller.

Function Name	Function Type	Type	Remarks
TextToValue	text command	Scr	Converts a string to a numerical point value.
UploadPLCProgram	PLC command	All	Uploads programs in the PLC to specified files.
ValueToText	text command	Scr	Converts a numerical value into a text point.
vertical%fill	object command	OP	Specifies the vertical fill of an object.
ViewReport	report command	All	Displays a report
visible	object command	OP	Toggles the visibility of an object.
width	object command	OP	Specifies the width of an object.
Write	file command	Scr	Writes a value to an open file.
WriteMessage	file command	All	Writes text to an external file.

The 'Type' column refers to the types of script and expression the function can be applied to. 'All' refers to both expressions and scripts. 'Scr' refers to scripts only. 'OP' refers to Object and Page scripts only.

Object Commands

Object commands control native CX-Supervisor graphical objects, like rectangles or lines.

Note: Objects are native to CX-Supervisor and therefore cannot be accessed or commands issued from external script languages, like VBScript or Jscript.

Current Object

Syntax

```
objectcommand
```

Remarks

Argument	Description
objectcommand	• The expression can be made up of the following commands, which are also described in chapter 6, Object Commands: • Colour command. • Disable command. • Visible command. • Move command. • Rotate command. • Vertical fill command. • Horizontal fill command. • Height command. • Width command. The content of the commands are made up of arithmetical or logical expressions, *x* and *y* co-ordinates, or references, varying between commands. The colour command requires a colour identifier.

Typical Example

```
colour (red)
```

The current object is specified as red in colour.

References

Refer to:

- Chapter 6, Blink for use of the blink command.
- Chapter 6, Colour for use of the colour command.
- Chapter 6, Disable for use of the disable command.
- Chapter 6, Height for use of the height command.
- Chapter 6, Horizontal *Fill* for use of the horizontal fill command.
- Chapter 6, Move for use of the move command.
- Chapter 6, Rotate for use of the rotate command.
- Chapter 6,Vertical *Fill* for use of the vertical fill command.
- Chapter 6, Visible for use of the visible command.
- Chapter 6, Width for use of the width command.
- *The CX-Supervisor User Manual* for details of the Animation Editor.

Other Objects

Syntax

```
objectname.objectcommand
```

```
pagename.objectname.objectcommand
```

Remarks

Argument	Description
objectname	This is the name of the object. The object is provided with a generic name on creation, which can be amended later to something more meaningful. The script is automatically updated following any amendment to the object name.
objectcommand	This can be made up of the following commands, which are described in chapter 6, Object Commends: • Blink command • Colour command. • Disable command. • Visible command. • Move command. • Rotate command. • Vertical fill command. • Horizontal fill command. • Height command. • Width command. The content of the commands are made up arithmetical or logical expressions, *x* and *y* co-ordinates, or references, varying between commands. The colour command requires a colour identifier.

Typical Examples

```
POLYGON_1.colour (red)
POLYGON_1.colour = red
```

The specified object, 'POLYGON_1' is set to be red in colour.

References

Refer to:

- *CX-Supervisor User Manual* for details of object names.
- Chapter 6, Blink for use of the blink command.
- Chapter 6, Colour for use of the colour command.
- Chapter 6, Disable for use of the disable command.
- Chapter 6, Height for use of the height command.
- Chapter 6, Horizontal Fill for use of the horizontal fill command.
- Chapter 6, Move for use of the move command.
- Chapter 6, Rotate for use of the rotate command.

- Chapter 6, Vertical Fill for use of the vertical fill command.
- Chapter 6, Visible for use of the visible command.
- Chapter 6, Width for use of the width command.

Blink

Syntax

```
objectname.blink (colour, status)
```

Remarks

Argument	Description
objectname	This is the name of the object. Where a script is directly attached to an object, *objectname* is not required.
colour	Colour to blink to. Some colour values within the colour palette have a meaningful *colourID*. This takes the form of the colour name, e.g., 'black' or 'yellow'. Alternatively, an integer value of 0x1000000 can be added to a number 0-65 to select a palette entry.
status	This argument may be omitted. May be on of: TRUE – turn blinking On. FALSE – turn blinking Off. If omitted, TRUE is assumed.

Typical Examples

```
blink (red, TRUE)
```

Start blinking red.

```
LINE_1.blink(OxFFFFOO, status)
```

The object LINE_1 starts or stops blinking yellow depending on value of Boolean point 'status'.

Colour

Syntax

```
objectname.colour (expression, context)
colour (expression, context)
```

or

```
objectname.colour (colourID, context)
colour (colourID, context)
An equals sign may be used as an alternative to brackets:
objectname.colour = expression
colour = expression
```

or

```
objectname.colour = colourID
```

```
colour = expression
```

Either spelling 'colour' or 'color' is acceptable.

Note: An equals sign may also be used for most other object commands, even if it is not directly specified in this manual.

Remarks

Argument	Description
objectname	This is the name of the object. Where a script is directly attached to an object, *objectname* is not required.
expression	The *expression* may be an Integer point, or a calculation of constants and/or points that produce an Integer value between 0 and 16777215. This is the desired colour's RGB value. (format is 0xBBGGRR).
colourID	Some colour values within the colour palette have a meaningful *colourID*. This takes the form of the colour name, e.g., 'black' or 'yellow'. Alternatively, an integer value of 0x1000000 can be added to a number 0-65 to select a palette entry.
context	This argument is optional an may be omitted. It defines which part of the object has it's colour changed. May be one or more of: @FILL – change fill colour @FRAME – changes frame colour If omitted both are changed. Equivalent to @FILL | @FRAME

Typical Examples

```
TEXT_3.colour (blue)
```

or

```
TEXT_3.colour = blue
```

The object 'TEXT_3' is set to blue.

```
BALL.colour (35 + 0x1000000)
```

The object 'BALL' is set to colour 35 from the colour palette.

```
BALL.colour (0xFF0000,@FILL)
```

The object 'BALL' is set to blue.

```
shade = tint1 + tint2
IF shade > 65 OR shade < 0 THEN
  shade = 0
ENDIF
ELLIPSE_1.colour (shade + 0x1000000)
```

The point 'shade' is set to a value based on 'tint1' and 'tint2', and is tested first to ensure that it is a value between 0 and 65. If 'shade' falls outside this range, then it cannot be applied as a colour to an

object, and is therefore reset to 0 (or black). ELLIPSE_1' is set to the palette colour of the value of shade.

References

Refer to chapter 6, Colour Palette for details of colour names and colour numbers.

Disable

Syntax

```
objectname.disable (expression)
```

Remarks

Argument	Description
objectname	This is the name of the selectable object. Where a script is directly attached to an object, *objectname* is not required.
expression	The expression can be made up of points resulting in 'TRUE' or 'FALSE'.

Typical Examples

```
disable (TRUE)
```

The current pushbutton object to which this example applies is disabled.

```
PUSH_8.disable (count AND flag)
```

The selectable object 'PUSH_8' is disabled provided Integer point 'count' AND Boolean point 'flag' return ''TRUE''.

Height

Syntax

```
objectname.height (expression, context)
objectname.height = expression
```

Remarks

Argument	Description
objectname	This is the name of the object, following any amendment to the object name. Where a script is directly attached to an object, *objectname* is not required.
expression	This is a value, point or an arithmetic expression returning a new height value in pixels.
context	This argument is optional and may be omitted. It defines which part of the object is the datum, and remains static. May be one of: @TOP – uses object top as datum. @CENTRE – uses object centre as datum @BOTTOM – uses object bottom as datum If omitted @CENTRE is assumed

Typical Examples

```
height (100)
or
height = 100
```

The height of the current object is set to 100.

```
LINE_1.height (stretch/offset, @top)
```

The height of object 'LINE_1' is changed to the value calculated by points 'stretch' and 'offset', keeping the top where it is.

Horizontal Fill

Syntax

```
objectname.horizontal%fill (expression, context)
```

Remarks

Argument	Description
objectname	This is the name of the object. Where a script is directly attached to an object, *objectname* is not required.
expression	This is an arithmetic expression that must return a value between 0 and 100. On return of a valid result, the fill commences from left to right.
context	This argument is optional and may be omitted. It defines which side of the object is filled from. May be one of: @LEFT – fill from the left @RIGHT – fill from the right If omitted, @LEFT is assumed

Typical Examples

```
horizontal%fill (50)
```

The current object to which this example applies is filled by 50%.

```
ELLIPSE_1.horizontal%fill (GAS_LEVEL, @RIGHT)
```

The object 'ELLIPSE_1' is filled from the right, provided the point 'GAS_LEVEL' returns a valid result, between 0 and 100.

Move

Syntax

```
objectname.move (x co-ordinate, y co-ordinate)
```

Remarks

Argument	Description
objectname	This is the name of the object. following any amendment to the object name. Where a script is directly attached to an object, *objectname* is not required.
x co-ordinate *y co-ordinate*	The *x* and *y* co-ordinates of the origin of the object at its resultant position in pixels are specified in the form (x, y). Points alone or as part of an arithmetic expression may be used as a basis for this expression.

Typical Examples

```
move (100, 200)
```

The current object to which this example applies is moved to the specified position.

```
POLYGON_1.move (xpos, ypos/5)
```

The object 'POLYGON_1' is moved to the position specified by points 'xpos' and 'ypos' divided by 5.

Rotate

Syntax

```
objectname.rotate (angle, context, fixed, xcoord, ycoord)
```

Remarks

Argument	Description
objectname	This is the name of the object. Where a script is directly attached to an object, *objectname* is not required.
angle	The angle of rotation can range between 0 to 360 in a clockwise direction. Points alone, or as part of an arithmetic expression may be used as an angle.
context	This argument is not required and may be omitted. May be one of: @TOPLEFT – rotate around top left of object @TOPCENTRE –rotate around top centre of object @TOPRIGHT – rotate around top right of object @CENTRELEFT – rotate around centre left of object @CENTRE – rotate around centre of object @CENTRERIGHT – rotate around centre right of object @BOTTOMLEFT – rotate around bottom left of object @BOTTEMCENTRE – rotate around bottom centre of object @ BOTTOMRIGHT – rotate around bottom right of object @USERDEFINED – user defined point specified in xcoord and ycoord.
fixed	This argument may be omitted. If this boolean value is true, the rotation origin is fixed to the screen, even if the object is moved. Otherwise, the rotation origin is relative to object position.
xcoord *ycoord*	Only required if @USERDEFINED is specified. These integer variables specify the rotation origin in pixels

Typical Examples

```
rotate (45)
```

The current object to which this example applies is rotated by 45°.

```
RECTANGLE_1.rotate(tilt, @USERDEFINED, 0, -100, 10)
```

The object 'RECTANGLE_1' is rotated by the value of 'tilt', about a point –100, 10 relative to the objects current position.

```
rotate (a * sin(b))
```

The current object is rotated based on the result of an arithmetic expression involving points named 'a and 'b'.

Vertical Fill

Syntax

```
objectname.vertical%fill (expression, context)
```

Remarks

Argument	Description
objectname	This is the name of the object. Where a script is directly attached to an object, *objectname* is not required.
expression	This is an arithmetic expression that must return a value between 0 and 100. On return of a valid result, the fill commences from bottom to top.
context	This argument may be omitted. May be one of: @DOWN – Fill object downwards @UP – Fill object upwards If omitted, @UP is assumed

Typical Examples

```
vertical%fill (50)
```

The current object to which this example applies is filled by 50%.

```
ELLIPSE_1.vertical%fill (OIL_QUANTITY, @DOWN)
```

The object 'ELLIPSE_1' is filled provided the point 'OIL QUANTITY' returns a valid result, between 0 and 100.

Visible

Syntax

```
objectname.visible (expression)
```

Remarks

Argument	Description
objectname	This is the name of the object. Where a script is directly attached to an object, *objectname* is not required.
expression	The expression can be made up of points resulting in 'TRUE' or 'FALSE'.

Typical Examples

```
visible (TRUE)
```

The current object to which this example applies becomes visible.

```
POLYLINE_8.visible (count AND flag)
```

The object 'POLYLINE_8' is made visible provided Integer point 'count' AND Boolean point 'flag' return ''TRUE''.

Width

Syntax

```
objectname.width (expression, context)
```

Remarks

Argument	Description
objectname	This is the name of the object. Where a script is directly attached to an object, *objectname* is not required.
expression	This is a value, point or an arithmetic expression returning a new width value in pixels.
context	This argument may be omitted. May be one of: @LEFT – use left of object as datum. @CENTRE – use centre of object as datum. @RIGHT – use right of object as datum. If omitted, @CENTRE is assumed.

Typical Examples

```
width (150)
```

The width of the current object is set to 150.

```
LINE_1.width (squeeze/offset, @RIGHT)
```

The width of object 'LINE_1' is changed to the value calculated by points 'squeeze' and 'offset', keeping the rightmost point fixed.

Page Commands

Display Page

Syntax

```
display ("pagename")
```

Remarks

Argument	Description
pagename	This is the name of the page for display, based on its filename without the file extension, e.g. the *pagename* for CAR.PAG is simply 'CAR'.

Typical Examples

```
display ("CAR")
```

The page 'CAR.PAG' is displayed.

```
textpoint = "CAR"
display(textpoint)
```

The page 'CAR.PAG' is displayed.

Close Page

Syntax

```
close ("pagename")
```

Remarks

Argument	Description
pagename	This is the name of the page for closure, based on its filename without the file extension, e.g. the *pagename* for CAR.PAG is simply 'CAR'. The *pagename* for closure must be currently open.

Note: The 'close' operation will cause the page to be unloaded, including all objects, ActiveX controls and scripts. Care must be taken not to attempt to access them after the close instruction.

Note: Where the script containing the 'close' instruction is on the page to be closed, this should be the last instruction in the script as it will cause the script to be unloaded.

Typical Examples

```
close("CAR")
```

The page 'CAR.PAG' is closed.

```
textpoint = "CAR"
close(textpoint)
```

The page 'CAR.PAG' is closed.

General Commands

Exponential

Description

Mathematical function to calculate a value raised to a power.

Syntax

```
result = Exp (value, exponent)
```

Remarks

Argument	Type	Description
result	integer	Point name to receive returned result of *value* raised to the power of *exponent*.
value	integer	Number to raise.
exponent	integer	Power to raise *value* by.

Typical Example

```
MSBMask = Exp (2, 15)
```

In this example, 'MSBMask' is assigned the value 2^{15}, i.e. 32,768.

PlayOLE

Description

Initiate an OLE verb or 'method' on an OLE 2 object. The verb number is object dependent so refer to the object's documentation. This function is now largely obsolete as most objects are nowadays ActiveX objects.

Syntax

```
returnstate = PlayOLE("objectname",OLEVerbNumber)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
objectname	string	The identifier of the OLE object to be played.
OLEVerbNumber	integer	The verb number has a specific meaning to the OLE application. Typical values are: 0: specifies the action that occurs when an end-user double clicks the object in its container. The object determines this action (often 'edit' or 'play'). -1: instructs the object to show itself for editing or viewing. Usually an alias for some other object-defined verb. -2: instructs an object to open itself for editing in a window separate from that of its container. -3: causes an object to remove its user interface from the view. Applies only to objects that are activated in-place. Positive numbers designate object specific verbs.

Typical Example

```
PlayOLE("ole_1",0)
```

The object 'ole_1' is played using its primary verb.

PlaySound

Description

Plays a Windows .WAV sound file using the standard Windows sound channel and Sound Card driver.

Syntax

```
returnstate = PlaySound("soundfile")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
soundfile	string	Path of sound file to be played.

Typical Example

```
PlaySound("c:\noise.wav")
```

The soundfile "c:\noise.wav" is played.

Rand

Description

Returns a random integer, between 0 and the specified limit.

Syntax

```
pointname = Rand(upperlimit)
```

Remarks

Argument	Type	Description
upperlimit	integer	The maximum negative or positive integer value that the Rand function can generate.
pointname	Integer point	Point that contains the integer returned from the Rand function.

Typical Example

```
randomnumber = Rand(upperlimit)
```

A random integer in the range 0 to upperlimit is returned and contained in the point 'randomnumber'. Maximum upperlimit is 32767.

Note: If 'upperlimit' is negative then the range is 0 to the negative number.

RunApplication

Description

Requests the operating system runs a new program. It will run in a separate process and RunApplication does not wait for the application to be launched. The specified filename must be executable i.e. have an extension of .EXE, .COM or .BAT.

Syntax

```
returnstate = RunApplication("executable")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
executable	string	Pathname of executable file.

Typical Example

```
RunApplication("c:\myprog.exe")
```

The executable file c:\myprog.exe is run.

RunHelp

Description

Invokes the Windows Help engine and loads a help file, showing a specific topic number.

Syntax

```
returnstate = RunHelp("helpfile",helpindex)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
helpfile	string	Pathname of helpfile to be run.
helpindex	integer	Index into a help topic as defined by the help file being run.

Typical Example

```
RunHelp("c:\myhelp.hlp",0)
```

The helpfile c:\myhelp.hlp is run, and topic 0 shown.

SetLanguage

Description

Change the language of text on display. This will reload the system language file from the program folder (i.e. with a .LNG extension), and the user defined text from the application folder (i.e. with a .USL extension). This function is the programmatic equivalent of the user right clicking and changing the "Language Settings…" option.

Syntax

```
SetLanguage("language name")
```

Remarks

Argument	Type	Description
language name	string	Name of language to set to. Must be identical to filename of related file with ".lng" file extension. Standard options are English, Czech, Danish, Deutsch, Español, Finnish, French, Italiano, Nederlands (België), Norwegian, Português, Slovenija and Swedish. In addition "Default" will load the designers default language.

Typical Example

```
SetLanguage("Español")
```

In this example, the Spanish language files will be loaded.

```
SetLanguage("Default")
```

In this example, the language will revert to the default specified by the application designer.

ShutDown

Description

Closes the CX-Supervisor application.

Syntax

```
returnstate = ShutDown()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
ShutDown()
```

CX-Supervisor runtime operation is terminated.

Sleep

Description

Pause execution of a script for specified duration.

Syntax

```
Sleep (duration)
```

Remarks

Argument	Type	Description
Duration	---	Number of milliseconds to wait before continuing.

Typical Example

```
Sleep (1000)
```

CX-Supervisor waits 1 second.

Note 1: The sleep statement should be used with caution, as some other parts of the system may not be updated while a script is sleeping. It also uses multithreading which means some tasks like PLC communication may occur in parallel and behave unpredictably.

Note 2: In a well designed, truly event driven system use of the Sleep() statement should never be required. Always consider if the statements after the Sleep should be in their own script, executed when a Condition occurs.

Note 3: The Granularity (or intervals) differs between Operating Systems. In Windows NT (and 2000) expiration is checked every 10ms, so `'Sleep(100)'` actually pauses for any time between 100 to 109.99 milliseconds depending on when it was started. For Windows 98 (and ME) the granularity is 55ms so `'Sleep(100)'` actually pauses for 110 (2 times 55) to 164.99 milliseconds (nearly 3 times 55). For this reason, Sleep statements can act differently on different Operating Systems making the application OS dependant.

Note 4: Sleep should never be used as a delay for timing processes, for the following reasons:
- The actual time delay depends on the OS as described above
- There is always an error of 0 to 1 granularity, depending on when the action is started.
- The frequency can not be guaranteed as the OS may be busy, or handling other processes.

Communications Commands

CloseComponent

Syntax

```
Returnstate = CloseComponent(ComponentName, PLCName)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
ComponentName	text	A Text point or text constant containing the name of the component to close.
PLCName	text	Text point or text constant containing the name of the PLC that the component to close is attached to.

Typical Examples

```
CloseComponent("PLC Data Monitor", "MyPLC")
```

In this example, the PLC Data Monitor component monitoring the PLC 'MyPLC' is closed.

```
Component = "Performance Monitor"
PLC = "PLC06"
OK = CloseComponent(Component, PLC)
```

In this example, the Performance Monitor component monitoring the PLC 'PLC06' is closed. 'OK' is used to determine if the action was successful.

EnableOLE

Syntax

```
returnstate = EnableOLE(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
Pointname	bool point	A Boolean point that holds the required enable/disable state.

Typical Examples

```
EnableOLE(result)
```

OLE functions are enabled based on the value of point 'result'. If result is 'TRUE', then OLE is enabled. If result is 'FALSE', then OLE is disabled.

```
EnableOLE(TRUE)
```

OLE functions can also be enabled directly without using a point to hold the desired status.

EnablePLC

Syntax

```
returnstate = EnablePLC(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	bool point	A Boolean point that holds the required enable/disable state.

Typical Examples

```
EnablePLC(result)
```

PLC functions are enabled based on the value of point 'result'. If result is 'TRUE', then PLC functions are enabled. If result is 'FALSE', then they are disabled.

```
EnablePLC(TRUE)
```

PLC functions can also be enabled directly without using a point to hold the desired status.

OpenComponent

Syntax

```
Returnstate = OpenComponent(ComponentName, PLCName)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
ComponentName	text	A Text point or text constant containing the name of the component to open.
PLCName	text	Text point or text constant containing the name of the PLC that the component to open is attached to.

Typical Examples

```
OpenComponent("PLC Data Monitor", "MyPLC")
```

In this example, the PLC Data Monitor component monitoring the PLC 'MyPLC' is opened.

```
Component = "Performance Monitor"
PLC = "PLC06"
OK = OpenComponent(Component, PLC)
```

In this example, the Performance Monitor component monitoring the PLC 'PLC06' is opened. 'OK' is used to determine if the action was successful.

Point Commands

CancelForce

Syntax

```
returnstate = CancelForce(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point. If the point is an array point then all elements within the array have the CancelForce command applied.

Typical Example

```
CancelForce(point1)
```

The forcing of values on the point 'point1' is cancelled.

References

Refer to PLC operation manuals for a detailed description of Force Set, and Force Reset.

CopyArray

Syntax

```
CopyArray (SourceArray, DestArray)
```

Remarks

Argument	Type	Description
SourceArray	---	Name of point array to copy from.
DestArray	---	Name of point array to copy to.

Typical Example

```
InitArray (DestArray, 0)
```

First initialise 'DestArray'.

```
SourceArray [0] = 1
SourceArray [1] = 2
SourceArray [2] = 3
```

Then, initialise 'SourceArray' to {1, 2, 3}.

```
CopyArray (SourceArray, DestArray)
```

Finally, copy the content of the source array 'SourceArray' to the destination array 'DestArray'.

The two arrays do not have to be the same size as each other, for example if 'DestArray' contains 20 elements, only elements [0], [1] and [2] are set to 1, 2 and 3 respectively, the remaining elements are unchanged i.e. O's. If 'DestArray' is smaller than 'SourceArray' i.e. it contains two elements then only elements [0] and [1] are set to 1 and 2 respectively.

Note: 'CopyArray' accepts arrays of different type i.e. Boolean arrays can be copied into Real arrays, the only restriction is that Text arrays cannot be copied into numeric arrays and vice- versa.

DisableGroup

Syntax

```
returnstate = DisableGroup(groupname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
groupname	text	Name of the group containing the points to disable.

Typical Example

```
DisableGroup("<Default>")
```

All points belonging to the <Default> group is disabled thus preventing values from being read\written.

DisablePoint

Syntax

```
returnstate = DisablePoint(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
Pointname	point	Name of point to be disabled.

Typical Example

```
DisablePoint(point1)
```

The point 'point1' is disabled thus preventing values to be read/written.

Note: This is useful for optimisation of communications.

EditPoint

Syntax

```
EditPoint(BoolPoint, Caption, OffText, OnText)
```

or

```
EditPoint(AnalogPoint, Caption, MinValue, MaxValue, Keyboard)
```

or

```
EditPoint(TextPoint, EchoOff, Keyboard)
```

Remarks

Argument	Type	Description
BoolPoint	point	Name of Boolean point to be edited
Caption	Text	Text Caption for Edit dialog
OffText	Text	Text description for Boolean state 0
OnText	Text	Text description for Boolean state 1
AnalogPoint	point	Name of Integer or Real point to be edited
MinValue	Int/Real	Minimum value to be entered
MaxValue	Int/Real	Maximum value to be entered
Keyboard	Bool	Flag set to TRUE to display the onscreen keyboard
TextPoint	point	Name of Text point to be edited
EchoOff	Bool	Flag set to TRUE if input is not to be echoed for security

Typical Example

```
EditPoint(bFlag, "Select ON or OFF", "ON", "OFF")
```

A dialog is displayed to edit the Boolean point 'bFlag', to "ON" or "OFF" with a caption "Select ON or OFF".

```
EditPoint(nValue, "Enter a new value", 0.000000, 9999.000000, FALSE )
```

A dialog is displayed to edit the analogue point 'nValue', between 0 and 9999 with a caption "Enter a new value" without using the onscreen keyboard.

```
EditPoint(txtMessage, "Set Text to", FALSE ,FALSE )
```

A dialog is displayed to edit the Text point 'txtMessage', with a caption "Set Text to", echoing the input and not displaying the onscreen keyboard.

EnableGroup

Syntax

```
returnstate = EnableGroup(groupname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
groupname	text	Name of the group containing the points to enable.

Typical Example

```
EnableGroup("<Default>")
```

All points belonging to the '<Default>' group is enabled thus allowing values to be read\written.

EnablePoint

Syntax

```
returnstate = EnablePoint(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point to be enabled.

Typical Example

```
EnablePoint(point1)
```

The point 'point1' is enabled thus allowing values to be read/written.

Force

Syntax

```
returnstate = Force(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point to have force state applied. If the point is an array point then all elements within the array have the Force command applied.

Typical Example

```
Force(point1)
```

The point 'point1' is locked in its current state. i.e. if it is currently set to 1 it cannot be changed until the forced state is removed via the CancelForce command.

ForceReset

Syntax

```
returnstate = ForceReset(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point. If the point is an array point then all elements within the array have the ForceReset command applied.

Typical Example

```
ForceReset(point1)
```

The Boolean point 'point1' has its value set to 'FALSE'.

References

Refer to PLC operation manuals for a detailed description of ForceSet, and ForceReset.

ForceSet

Syntax

```
returnstate = ForceSet(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	point	Name of point. If the point is an array point then all elements within the array have the ForceSet command applied.

Typical Example

```
ForceSet(point1)
```

The Boolean point 'point1' has its value set to 'TRUE'.

References

Refer to PLC operation manuals for a detailed description of Force Set, and Force Reset.

GetBit

Syntax

```
returnpoint = GetBit(pointname,bit)
```

Remarks

Argument	Type	Description
pointname	Integer / real	This is the name of the point to get the bit value from. Indirection or point value may be used.
bit	integer	This specifies which bit to get the value of.
returnpoint	bool	This contains the return value 'TRUE' or 'FALSE'.

Typical Example

```
pointname = 256;
returnpoint = GetBit(pointname,8)
```

The point 'returnpoint' contains 'TRUE'.

GetPointValue

Syntax

```
returnpoint = GetPointValue(pointname,offset)
```

Remarks

Argument	Type	Description
pointname	point	This is the name of the point whose contents are to be returned.
offset	integer	This specifies the offset into an array point. 0 if the point is not an array point.
returnpoint	point	Point that contains the return value. The type of data returned is dependant on the pointname specified.

Typical Example

```
pointname = 10;
returnpoint = GetPointValue(pointname,0)
```

The point 'returnpoint' contains the value 10. The offset is added to any offset specified for pointname. For example:

```
returnpoint = GetPointValue(a[10],10)
```

Causes the 21st element (offsets begin at zero) of array 'a' to be retrieved.

Note: It is often simpler to access an array element directly, e.g. returnpoint = a[20].

InitialiseArray

Syntax

```
InitArray (arrayname, value)
```

Remarks

Argument	Type	Description
arrayname	---	Name of point array.
value	---	Value to set all elements of the array to.

Typical Example

```
InitArray (MyArray, 0)
```

In this example, all elements of the array 'MyArray' are set to 0.

InputPoint

Syntax

```
returnstate = InputPoint(pointname, returnflag)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	point	The point name whose data is to be read.
returnflag	point	Optional Boolean point which is set to 'TRUE' when value is returned from the PLC.

Typical Examples

```
InputPoint(point)

returnflag = FALSE
InputPoint(point, returnflag)
```

A request is made that the current value of point 'point' should be read. In the second example, returnflag is set to 'TRUE' when the value is returned from the PLC.

Note: The value is not returned immediately - it is not possible to use the returned value in the same script as the InputPoint command. Instead, the value should be accessed from within an "On Condition" script which has an expression of 'returnflag = TRUE'.

OutputPoint

Syntax

```
returnstate = OutputPoint(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	point	The point to be updated.

Typical Examples

```
OutputPoint(result)
```

The point 'result' is updated with its current value.

Note: The value of a point connected to a PLC is not be set if the point is currently in a "forced" state.

PointExists

Syntax

```
returnpoint = PointExists(pointname)
```

Remarks

Argument	Type	Description
pointname	string	This text contains the point name.
returnpoint	point	Boolean point that contains the return value.

Typical Example

```
PointName="Testpoint"
Exists=PointExists(PointName)
```

The Boolean point 'Exists' is set to 'TRUE' if a point called 'TestPoint' exists.

Note: "PointName" is a text point which can be set to any string value.

SetBit

Syntax

```
returnstate = SetBit(pointname,bit,value)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	integer/ real	This is the name of the point to set the bit for. Indirection or point arrays may be used.
bit	integer	This specifies the bit to set.
value	bool	This specifies the value to set the bit to.

Typical Example

```
testpoint = 0;
SetBit(testpoint,4,TRUE)
```

The point 'testpoint' contains the value 16.

SetPointValue

Syntax

```
returnstate = SetPointValue(pointname,offset,value)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	point	This is the name of the point whose contents are to be updated. Indirection or point arrays may be used.
offset	integer	This specifies the offset into an array point. 0 if the point is not an array point.
value	Integer / real	The new value that the chosen point is to be updated to.

Typical Example

```
SetPointValue(value,0,10)
```

The point 'value' contains the value 10.

Note: The value of a point connected to a PLC is not set if the point is currently in a "forced" state.

The offset is added to any offset specified for pointname. For instance:

```
SetPointValue(a[10],10,100)
```

Causes the 21st element (offsets begin at zero) of array 'a' to be set to 100. It is often simpler to access an array element directly, for instance:

```
a[20] = 100
```

PLC Commands

ClosePLC

Syntax

```
returnstate = ClosePLC("plcname")
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC to be opened. If the PLC is being accessed using a communications component, e.g. the Omron CX-Communications Control this parameter should be the control name and PLC name separated by a dot e.g. "OMRONCXCommunicationsControl.controlPLC".

Typical Example

```
ClosePLC("controlPLC")
```

The PLC called controlPLC is closed. No further communications with the PLC will take place until it is reopened.

DownloadPLCProgram

Syntax

```
returnstate = DownloadPLCProgram(plcname, filename, processed)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
plcname	string	Name of PLC to download the program to.
filename	string	Name of the file on disk to download to the PLC. If a drive and path are not specified, the current directory is assumed, which may not be the same as the application directory. If a filename is specified as "" the user is prompted at runtime for a filename.
processed	bool	*processed* is set to 'TRUE' when the operation is actually completed.

Typical Example

```
DownloadPLCProgram("controlPLC", "Prog01.bin", done)
DO UNTIL done or timeout > 5
  Sleep(10)
  timeout += 0.01
LOOP
```

The program stored in the file 'Prog01.bin' in the current directory is downloaded to the PLC 'controlPLC'. Before continuing, the script waits up to five seconds for the action to succeed.

Note: The operation may not be complete immediately after the statement has been executed. The processed flag 'done' is set at a later time when the operation has been completed. Therefore, if using statements that require the upload to be completed, either wait for the flag to be set (as shown) or create an On Condition script containing the code to be executed after the upload, with the processed flag as the expression (e.g. 'done').

Note: This command can only be used when the PLC is in 'STOP' mode. Refer to chapter 6, GetPLCMode or chapter 6, SetPLCMode for further information.

GetPLCMode

Syntax

```
mode = GetPLCMode("plcname")
```

Remarks

Argument	Type	Description
mode	string	A Text point containing the current PLC mode. Possible modes are 'STOP', 'DEBUG', 'RUN', 'MONITOR' and 'UNKNOWN'.
plcname	string	Name of the PLC.

Typical Example

```
currentmode = GetPLCMode("controlPLC")
```

In this example, the current mode of the PLC 'controlPLC' is stored in the point 'currentmode'.

OpenPLC

Syntax

```
Returnstate = OpenPLC("plcname", processed)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
plcname	string	Name of PLC to be opened. If the PLC is being accessed using a communications component, e.g. the Omron CX-Communications Control this parameter should be the control name and PLC name separated by a dot e.g. "OMRONCXCommunicationsControl.controlPLC".
processed	bool	Flag set to TRUE when set operation has actually been completed.

Typical Example

```
OpenPLC("controlPLC", doneopen
DO UNTIL doneopen or timeout > 5
      Sleep(10)
      timeout += 0.01
LOOP
```

The PLC called controlPLC is opened for communication.

Note that the PLC may not be opened immediately after the statement has been executed. The *processed* flag will be set at a later time when the operation has been completed. Therefore, if using statements which require the operation to be completed, either wait for the flag to be set as shown above, or create an On Condition script containing the code to be executed after the PLC is opened with the 'processed' flag as the expression (this is generally more efficient).

PLCCommsFailed

Syntax

```
returnstate = PLCCommsFailed("plcname")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
plcname	string	Name of PLC to be checked.

Typical Example

```
IsFailing = PLCCommsFailed ("controlPLC")
```

The point IsFailing is set to true if the PLC called controlPLC is currently not communicating. Otherwise it is set to false.

Note: This function returns to TRUE from the time when a communications timeout error with the named PLC occurs, until successful communication with the PLC takes place.

PLCMonitor

Syntax

```
returnstate = PLCMonitor("plcname")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
plcname	string	Name of PLC to be monitored.

Typical Example

```
PLCMonitor("controlPLC")
```

The monitor dialog for the PLC called controlPLC is invoked. This dialog can be used to check PLC status, change mode, etc.

SetPLCMode

Syntax

```
returnstate = SetPLCMode("plcname", mode, processed)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
plcname	string	Name of PLC.
mode	string	A value for the new PLC mode. Valid modes are 'STOP', 'DEBUG', 'RUN' and 'MONITOR'.
processed	bool	*processed* is set to 'TRUE' when the operation is actually completed.

Typical Examples

```
SetPLCMode("controlPLC", "STOP", done)
DO UNTIL done or timeout > 5
  Sleep(10)
  timeout += 0.01
LOOP
```

In this example, the mode of the PLC called 'controlPLC' is changed to "STOP". Before continuing, the script waits up to five seconds for the action to succeed.

Note: The mode may not be changed immediately after the statement has been executed. The processed flag 'done' is set at a later time when the operation has been completed. Therefore, if using statements that require the operation to be completed, either wait for the flag to be set (as shown) or create an On Condition script containing the code to be executed after the mode is set, with the processed flag as the expression (e.g. 'done').

SetPLCPhoneNumber

Syntax

```
Returnstate = SetPLCPhoneNumber("plcname", numbertext)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
plcname	string	Name of PLC to change the number of.
numbertext	string	New phone number for the PLC.

Typical Example

```
SetPLCPhoneNumber("controlPLC", "01234 987654")
```

The phone number for the PLC is changed to the required value.

UploadPLCProgram

Syntax

```
returnstate = UploadPLCProgram(plcname, filename, processed)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
plcname	string	Name of PLC to upload the program from.
filename	string	Name of the file on disk to upload the program to. If a drive and path are not specified, the file is created in the current directory, which may not be the same as the application directory. If a filename is specified as "" the user is prompted at runtime for a filename.
processed	bool	*processed* is set to 'TRUE' when the operation is actually completed.

Typical Example

```
UploadPLCProgram("controlPLC", "Prog01.bin", done)
DO UNTIL done or timeout > 5
  Sleep(10)
  timeout += 0.01
LOOP
```

The program in the PLC 'controlPLC' is uploaded to the file 'Prog01.bin' in the current directory. Before continuing, the script waits up to five seconds for the action to succeed.

Note: The operation may not be complete immediately after the statement has been executed. The processed flag 'done' is set at a later time when the operation has been completed. Therefore, if using statements that require the upload to be completed, either wait for the flag to be set (as shown) or create an On Condition script containing the code to be executed after the upload, with the processed flag as the expression (e.g. 'done').

Note: This command can only be used when the PLC is in 'STOP' mode. Refer to chapter 6, GetPLCMode or chapter 6, SetPLCMode for further information.

Temperature Controller Commands

TCAutoTune

Syntax

```
returnstate = TCAutoTune(TController,mode)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	string	This is a string representing the name of the temperature controller.
mode	point	This is a point depicting the mode of operation and defines the operation to be carried out when a TCAutoTune command is issued. 0: Indicates that the auto-tuning operation is to be stopped. 1: This mode is supported on the E5*K and is used to set the limit cycle of the manipulated variable change width to 40%. 2: This is used to start the auto-tuning operation.

Typical Example

```
temp1 = TCAutoTune("e5ak",temp2)
```

TCBackupMode

Syntax

```
returnstate = TCBackupMode(TController,mode)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	string	This is a string representing the name of the temperature controller.
mode	point	This is a point depicting the mode of operation and defines the method used by a temperature controller for storing internal variables. 0: In this mode variables are stored in RAM and EPROM. 1: In this mode variables are stored in RAM only.

Typical Example

```
temp1 = TCBackupMode("ea5k",temp2)
```

TCGetStatusParameter

Syntax

```
returnstate = TCGetStatusParameter(TController,paramID,value)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	string	This is a string representing the name of the temperature controller.
paramID	point	This is a point depicting the required parameter range 0 to 22: 0: ControlMode. 1: Output. 2: InputShiftDelay (Bool) E5*F, E5*X, E5*J. 3: DisplayUnit. 4: PIDConstantDisplay (Bool) E5*F, E5*X, E5*J. 5: OutputType. 6: CoolingType. 7: Output2. 8: Alarm1. 9: Alarm2. 10: InputType (Integer) E5*F, E5*X, E5*J. 11: OperationMode. 12: BackupMode. 13: AutoTuneMode. 14: OverFlow (Bool) E5*F, E5*X, E5*J. 15: UnderFlow (Bool) E5*F, E5*X, E5*J. 16: SensorMalfunction (Bool) E5*F, E5*X, E5*J. 17: ADConvertorFailure (Bool) E5*F, E5*X, E5*J. 18: RAMAbnormality (Bool) E5*F, E5*X, E5*J. 19: RAMMismatch (Bool) E5*F, E5*X, E5*J. 20: StatusWordsOnly (Bool) E5*K only (TRUE indicates valid words below). 21: Status0 (word) E5*K only. 22: Status1 (word) E5*K only.
value	point, real or int	The returned status parameter value. Refer to *paramID* above for details.

Typical Example

```
temp1 = TcGetStatusParameter("e5ak",temp2,temp3)
```

TCRemoteLocal

Syntax

```
returnstate = TCRemoteLocal(TController,mode)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	string	This is a string representing the name of the temperature controller.
mode	point	This is a point depicting the mode of operation and defines the operational mode of a temperature controller. 0: This specifies the temperature controller is in remote mode. 1: This specifies that the temperature controller is in local mode.

Typical Example

```
temp1 = TCRemoteLocal("e5ak",temp2)
```

Note: This command was previously called TCOperationalMode.

TCRequestStatus

Syntax

```
returnstate = TCRequestStatus(Tcontroller, returnflag)
```

Remarks

Argument	Type	Description
returnstate	Bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	String	This is a string representing the name of the temperature controller.
returnflag	Point	This is a point depicting that the status has been returned and is available for the command TCGetStatusParameter.

Typical Example

```
temp1 = TCRequestStatus("e5ak", temp2)
```

Note: The status information is NOT returned immediately - it is not possible to access the status information in the same script as the TCRequestStatus command. Instead, the status information should be accessed from within an "On Condition" script which has an expression of "returnflag == TRUE".

TCRspLsp

Syntax

```
returnstate = TCRspLsp(Tcontroller,mode)
```

Remarks

Argument	Type	Description
returnstate	Bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	String	This is a string representing the name of the temperature controller.
mode	Point	This is a point depicting the mode of operation and defines the setpoint mode used by the temperature controller. 0: This specifies remote setpoint mode. 1: This specifies local setpoint mode.

Typical Example

```
temp1 = TCRspLsp("e5ak",temp2)
```

Note: This command was previously called TCSetpoint.

TCRunStop

Syntax

```
returnstate = TCRunStop(TController,mode)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	string	This is a string representing the name of the temperature controller.
mode	point	This is a point depicting the mode of operation and defines either auto-output mode shift or manual output mode shift. 0: This specifies manual output mode shift. 1: This specifies auto-output mode shift.

Typical Example

```
temp1 = TCRunStop("e5ak",temp2)
```

Note: This command was previously called TCModeShift.

TCSaveData

Syntax

```
returnstate = TCSaveData(TController)
```

Remarks

Argument	Type	Description
returnstate	Bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	String	This is a string representing the name of the temperature controller.

Typical Example

```
temp1 = TCSaveData("e5ak",temp2)
```

TCSettingLevel1

Syntax

```
returnstate = TCSettingLevel1(TController)
```

Remarks

Argument	Type	Description
returnstate	Bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	String	This is a string representing the name of the temperature controller.

Typical Example

```
temp1 = TCSettingLevel1("e5ak")
```

TCReset

Syntax

```
returnstate = TCReset(TController)
```

Remarks

Argument	Type	Description
returnstate	Bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
TController	String	This is a string representing the name of the temperature controller.

Typical Example

```
temp1 = TCReset("e5ak")
```

Alarm Commands

AcknowledgeAlarm

Syntax

```
returnstate = AcknowledgeAlarm("alarmname")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
alarmname	string	This is the identifier of the alarm.

Typical Example

```
AcknowledgeAlarm("temphigh")
```

The alarm 'temphigh' is acknowledged.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

AcknowledgeAllAlarms

Syntax

```
returnstate = AcknowledgeAllAlarms()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
AcknowledgeAllAlarms()
```

All alarms are acknowledged.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

AcknowledgeLatestAlarm

Syntax

```
returnstate = AcknowledgeLatestAlarm()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
AcknowledgeLatestAlarm()
```

The most current alarm of the highest priority is acknowledged.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

ClearAlarmHistory

Syntax

```
returnstate = ClearAlarmHistory()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
ClearAlarmHistory()
```

The alarm history window is cleared and the log is cleared.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

CloseAlarmHistory

Syntax

```
returnstate = CloseAlarmHistory()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
CloseAlarmHistory()
```

The alarm history window is closed.

References

Refer to the *CX-Supervisor User Manual* for details of alarms

CloseAlarmStatus

Syntax

```
returnstate = CloseAlarmStatus()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
CloseAlarmStatus()
```

The current alarm status window is closed.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

DisplayAlarmHistory

Syntax

```
returnstate = DisplayAlarmHistory()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
DisplayAlarmHistory()
```

The alarm history window is displayed.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

DisplayAlarmStatus

Syntax

```
returnstate = DisplayAlarmStatus()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
DisplayAlarmStatus()
```

The current alarm status is displayed.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

EnableAlarms

Syntax

```
EnableAlarms (flag, "message")
```

Remarks

Argument	Type	Description
flag	---	If set 'TRUE' then alarm logging is enabled. If set 'FALSE' logging is disabled.
message	---	Text message which is recorded in the alarm log to indicate change of status.

Typical Example

```
EnableAlarms (TRUE, "Alarm logging enabled")
```

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

IsAlarmAcknowledged

Syntax

```
pointname = IsAlarmAcknowledged("alarmname")
```

Remarks

Argument	Type	Description
pointname	bool point	The Boolean point name to be assigned a value based on the test of an acknowledged alarm.
alarmname	string	The identifier of the alarm.

Typical Example

```
acknowledged = IsAlarmAcknowledged("temptoohigh")
```

The point 'acknowledged' is assigned Boolean state ''TRUE'' if the 'temptoohigh' alarm is currently acknowledged. The point is assigned Boolean state 'FALSE' if the alarm is not currently acknowledged.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

IsAlarmActive

Syntax

```
pointname = IsAlarmActive("alarmname")
```

Remarks

Argument	Type	Description
pointname	bool point	The Boolean point name to be assigned a value based on the test of an active alarm.
alarmname	string	The identifier of the alarm.

Typical Example

```
active = IsAlarmActive("temptoohigh")
```

The point 'active' is assigned Boolean state ''TRUE'' if the 'temptoohigh' alarm is currently active. The point is assigned Boolean state 'FALSE' if the alarm is not currently active.

References

Refer to the *CX-Supervisor User Manual* for details of alarms.

File Commands

CloseFile

Syntax

```
returnstate = CloseFile(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	bool	A Boolean point that holds the required status of whether blank spaces should be stripped from the file when it is closed.

Typical Examples

```
CloseFile(status)
```

The currently open file is closed. Blank spaces at the end of each line are stripped from the file if the Boolean point 'status' is set to 'TRUE'.

```
CloseFile(FALSE)
```

In this example, the currently open file is closed and any blank spaces are not stripped from the file.

Note: If blank spaces are stripped from the file, then it greatly reduces in size but it takes slightly longer to close. Blank spaces should not be stripped from the file if it is being used on a network drive by more than one system at a time.

CopyFile

Syntax

```
returnstate = CopyFile("sourcename", "destname")
```

Remarks

Argument	Type	Description
Returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise
sourcename	string	Pathname of file to be copied. May include a "*" wildcard character.
destname	string	Pathname of destination of copy. If path name does not exist it is created.

Typical Example

```
CopyFile("c:\autoexec.bat", "c:\autoexec.old")
```

The file "c:\autoexec.bat" is copied to the file "c:\autoexec.old".

```
CopyFile("c:\logging\*.dlv", "a:\backup")
```

The data log files (ending in dlv) in "C:\logging" are copied to the "\backup" directory on drive A:

DeleteFile

Syntax

```
returnstate = DeleteFile("filename")
```

Remarks

Argument	Type	Description
Returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be deleted.

Typical Example

```
DeleteFile("c:\pagename.pag")
```

The file "c:\pagename.pag" is deleted.

EditFile

Syntax

```
returnstate = EditFile("filename")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
filename	string	Pathname of the file to be edited.

Typical Example

```
EditFile("C:\report3.txt")
```

FileExists

Syntax

```
returnpoint = FileExists (filename)
```

Remarks

Argument	Type	Description
filename	string	This text string contains the file name.
returnpoint	point	Boolean point that contains the return value.

Typical Example

```
FileName = "TEST.TXT"
Exists = FileExists(FileName)
```

The Boolean point 'Exists' is set to 'TRUE' if a file called 'C:\TEST.TXT' exists.

Note: "FileName" is a text point which can be set to any string value.

MoveFile

Syntax

```
returnstate = MoveFile("sourcename", "destname")
```

Remarks

Argument	Type	Description
Returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
sourcename	string	Pathname of file to be moved.
destname	string	Pathname of destination of move.

Typical Example

```
MoveFile("c:\autoexec.bat", "c:\autoexec.old")
```

The file "c:\autoexec.bat" is moved to the file "c:\autoexec.old".

OpenFile

Syntax

```
returnstate = OpenFile("filename")
```

Remarks

Argument	Type	Description
Returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be opened.

Typical Example

```
OpenFile("c:\filename")
```

The file "c:\filename.csf" is opened and able to be accessed by the Read() and Write() script commands. Only one file can be open at a time. Files can be shared (for instance located on a network drive, and accessed by several running CX-Supervisor applications simultaneously - this can be used for data exchange).

Note: An extension ".csf" will always be added to the filename so it must not be specifed as part of the argument.

PrintFile

Syntax

```
returnstate = PrintFile("filename")
```

Remarks

Argument	Type	Description
Returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be printed.

Typical Example

```
PrintFile("c:\autoexec.bat")
```

The file "c:\autoexec.bat" is sent to the currently configured printer.

Script commands that have textual arguments can take either literal strings within quotes or text points.

Note: CX-Supervisor uses the OLE registration information (file extension associations) to decide how to print a file. It invokes the parent application associated with a particular file extension, instructing the application to start minimised and passing the "print" command. For example, if the file extension .txt is associated with Notepad, then Notepad is invoked to print the file.

Read

Syntax

```
returnstate = Read(RecordId, pointname, ...)
```

Remarks

Argument	Type	Description
Returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
RecordId	integer	An index into the file.
Pointname	point	Name(s) of point(s) to be updated with the data read from the open file.

Typical Examples

```
Read(1, value)
```

The point 'value' is loaded with the value read from the currently open file using the value of 1 as an index into the file.

```
ReadOK = Read(indexno, value1, value2, value3)
```

The points 'value1', 'value2', 'value' are loaded using the value of indexno as an index into the file. Pass or fail status is stored in 'ReadOK'.

Note: It is advisable to use a RecordId less than 1024 whenever possible, in order to optimise file access time (records 0 to 1023 are cached).

ReadMessage

Syntax

```
returnstate = ReadMessage ("filename", offset, textpoint, noofchars)
```

Remarks

Argument	Type	Description
Returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
Filename	string	Pathname of file to be read.
Offset	integer	An offset from the beginning of the file (in characters) indicating where to start reading from.
Textpoint	text point	The text point which holds the characters read from the file.
Noofchars	integer	The number of characters to read from the file.

Typical Example

```
ReadMessage ("C:\CX-SUPERVISOR\TESTFILE.TXT", 0, TextPoint, 20)
```

The first 20 characters are be read from the file "C:\CX-SUPERVISOR\TESTFILE.TXT" and stored in the point 'TextPoint'.

Note: Text points can hold up to 256 characters therefore a maximum of 256 characters can be read from the file.

SelectFile

Syntax

```
filename = SelectFile (filter, path)
```

Remarks

Argument	Type	Description
Filename	---	Text string returned. Contains fully qualified filename including drive and path if OK was selected from OpenFile comms dialog, otherwise contains empty string.
Filter	string	Optional argument. If omitted, will show all files. This argument must be supplied if *path* is specified i.e. set to "". Specifies the filter string used by the 'Files of type' list. The string should contain 1 or more filters separated with a '|' (pipe) character and end with 2 characters i.e. '||'. Each filter should have some user text and 1 or more file specs separated with a semicolon. No spaces should be used, except within the user text.
Path	string	Optional argument. Specifies the path to show initially. If omitted, the dialog shows the current working directory.

Typical Example

```
TFile = SelectFile()
```

The 'File Open' dialog will be displayed, showing all files in the current working directory. The users choice will be stored in tFile.

```
TFile = SelectFile("Text Files (*.txt)|*.txt||")
```

The 'File Open' dialog will be displayed, showing just files with a .txt extension in the current working directory.

```
TFile = SelectFile("Text Files (*.txt; *.csv)|*.txt;*.csv||")
```

The 'File Open' dialog will be displayed, showing files with either a .txt or .csv extension in the current working directory.

```
TFile = SelectFile("Text Files (*.txt;
*.csv)|*.txt;*.csv|Document Files (*.doc)|*.doc||")
```

In this example, the 'Files of type' filter has 2 choices: one to show text files (i.e. both .txt and .csv files), and one to show document files (just .doc files).

```
TFile = SelectFile("", "C:\WINDOWS")
```

The 'File Open' dialog will be displayed, showing all files in the "C:\WINDOWS" directory.

Write

Syntax

```
returnstate = Write(RecordId, pointname, ...)
```

Remarks

Argument	Type	Description
Returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
RecordId	integer	An index into the file.
Pointname	point	Name(s) of point(s) containing data to write to the open file.

Typical Examples

```
WroteOK = Write(indexno, $Second)
```

The point '$Second' is written to the currently open file using the value of indexno as an index into the file. Pass or fail status is stored in 'WroteOK'.

```
Write(2, $Second, $Minute, $Hour)
```

The points '$Second', '$Minute', '$Hour' are written to the currently open file using the value 2 as an index into the file.

Note: It is advisable to use a RecordId less than 1024 whenever possible, in order to optimise file access time (records 0 to 1023 are cached).

WriteMessage

Syntax

```
returnstate = WriteMessage("filename", offset, "text", linefeed)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
filename	string	Pathname of file to be written.
offset	integer	An offset from the beginning of the file (in characters) indicating where to start writing. If the offset is -1 then the message is appended to the end of the file.
text	string	The text to be written into the file.
linefeed	bool	A flag to indicate a carriage return and line feed should be appended.

Typical Example

```
WriteMessage("C:\CX-SUPERVISOR\TESTFILE.TXT", 0, "Hello World", TRUE)
```

The text 'Hello World' is written at the start of the 'C:\CX-SUPERVISOR\TESTFILE.TXT' file and a carriage return and line feed is appended which moves and subsequent text to the start of the next line.

Note: When the text is written into the file it overwrites any existing text that may exist at this location.

Graph Commands

ClearGraph

Syntax

```
returnstate = ClearGraph("graphid", "pagename")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to be cleared.
pagename	string	Optional parameter indicating the name of the page that the graph is on.

Typical Examples

```
ClearGraph("Graph_1", "TestPage1")
```

The trend or scatter graph on 'TestPage1' with the identifier 'Graph_1' has its data cleared.

```
ClearGraph ("Graph_2")
```

The trend or scatter graph on the current page, with the identifier 'Graph_2', has its data cleared.

StartGraph

Syntax

```
returnstate = StartGraph("graphid", "pagename")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to be started.
pagename	string	*Optional* parameter indicating the name of the page that the graph is on.

Typical Examples

```
StartGraph("Graph_1", "TestPage1")
```

The trend or scatter graph on 'TestPage1' with the identifier 'Graph_1' has its data logging started.

```
StartGraph("Graph_2")
```

The trend or scatter graph on the current page with the identifier 'Graph_2' has its data logging started.

Note: This command is provided for compatibility with SCS v2.0 applications. For newer applications the data logging facilities should be used in preference.

StopGraph

Syntax

```
returnstate = StopGraph("graphid", "pagename")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend or scatter graph to be stopped.
pagename	string	*Optional* parameter indicating the name of the page that the graph is on.

Typical Examples

```
StopGraph("Graph_1", "TestPage1")
```

The trend or scatter graph on 'TestPage1' with the identifier 'Graph_1' has its data logging stopped.

```
StopGraph("Graph_2")
```

The trend or scatter graph on the current page with the identifier 'Graph_2' has its data logging stopped.

Recipe Commands

DisplayRecipes

Syntax

```
returnstate = DisplayRecipes()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
DisplayRecipes()
```

The current recipes is displayed.

References

Refer to the *CX-Supervisor User Manual* for details of recipes.

DownloadRecipe

Syntax

```
returnstate = DownloadRecipe("recipename")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
recipename	string	The name of the recipe to be downloaded.

Typical Example

```
DownloadRecipe("recipe1")
```

The recipe 'recipe1' is downloaded.

References

Refer to the *CX-Supervisor User Manual* for details of recipes.

UploadRecipe

Syntax

```
returnstate = UploadRecipe("recipename", processed)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
recipename	string	The name of the recipe to be uploaded.
processed	bool	Flag set to true when operation has been completed.

Typical Example

```
UploadRecipe("recipe1",done)
```

The recipe 'recipe1' is uploaded, and point 'done' is set True when the upload is complete.

References

Refer to the *CX-Supervisor User Manual* for details of recipes.

Report Commands

GenerateReport

Syntax

```
returnstate =
  GenerateReport(ReportTemplateFile,ReportOutputFile)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
ReportTemplateFile	string	Pathname of the report template file.
ReportOutputFile	string	Pathname of the report output file.

Typical Example

```
GenerateReport("report3.txt","output.txt")
```

PrintReport

Syntax

```
returnstate = Printreport(ReportTemplateFile)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
ReportTemplateFile	string	Pathname of the report template file.

Typical Example

```
PrintReport("report3.txt")
```

The ReportTemplateFile report3.txt contains a predefined set of point names and text laid out exactly as the report reader likes to view them. The point names contained within enclosing characters are the CX-Supervisor names for the data that is required in the report.

The enclosing characters can be changed in the Project/Runtime Setting/Report setting dialog box, but once set must be fixed for all reports generated by the project.

The file can be written using any ASCII text editor, for instance a Text file (.TXT), a Rich Text file (.RTF) or a Hypertext file (.HTML) .

Examples of the way in which text is changed:

```
The shift foreman was ((ShiftForeman))
```

Becomes:

```
The shift foreman was Fred Smith
The shift output was ((ShiftOutput)) litres
```

Becomes:

```
The shift output was 5000 litres
```

Format specifiers can be used:

```
%s for text string points;
%d for integer points;
%f for real floating points.
```

Where 'MyTextpoint="Hello"':

```
The text of (("My text point is %s",Mypoint))
```

Becomes:

```
The text of My text point is Hello
```

Where MyRealpoint=5467.7658:

```
To two decimal places the value of MyRealpoint is
(("%4.2f",MyRealpoint))
```

Becomes:

```
To two decimal places the value of MyRealpoint is 5467.76
```

Combinations of format specifiers are possible:

```
((%s to two decimal places the value of MyRealpoint is
%4.2f",MyTextpoint,MyRealpoint))
```

Becomes:

```
Hello to two decimal places the value of MyRealpoint is  5467.76
```

ViewReport

Syntax

```
returnstate = ViewReport(ReportTemplateFile)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
ReportTemplateFile	string	Pathname of the report template file.

Typical Example

```
ViewReport("report3.txt")
```

Text Commands

BCD

Syntax

```
result = BCD (value)
```

Remarks

Argument	Type	Description
Value	---	Number to convert to Binary Coded Decimal (BCD).
result	---	String containing BCD representation of *value*.

Typical Example

```
BCDStr = BCD(39)
```

In this example, 'BCDstr' contains '00111001'.

Bin

Syntax

```
result = Bin (value)
```

Remarks

Argument	Type	Description
value	---	Number to be converted to a binary number.
result	---	String containing binary representation of *value*.

Typical Example

```
BStr = Bin (20)
```

In this example, 'Bstr' contains '10100'.

Chr

Syntax

```
result = Chr (value)
```

Remarks

Argument	Type	Description
value	- - -	Extended ASCII value to convert to a character.
result	- - -	String containing single character representation of *value*.

Typical Example

```
Char = Chr(65)
```

In this example, 'Char' contains 'A'.

FormatText

Syntax

```
textpoint = FormatText ("formattext", expression, ...)
```

Remarks

Argument	Type	Description
textpoint	text point	A text point which holds the formatted text.
formattext	string	The text (with appropriate formatting characters) that the result *expression* is inserted into.
expression	Integer / real	The value(s) or expression(s) that is inserted into *formattext*.

Typical Examples

```
TextPoint = FormatText ("Boiler temperature is %ld degrees.",
  BoilerTemp)
```

The value of the 'BoilerTemp' point is inserted into the specified text at the position marked by the formatting characters (%ld) and then stored in the point 'TextPoint'.

If the value of 'BoilerTemp' was 57 then the resultant text that is stored in 'TextPoint' is as follows:

```
"Boiler temperature is 57 degrees."
```

```
TextPoint = FormatText ("Boiler %ld temperature is %ld degrees.",
  BoilerNo, BoilerTemp)
```

The value of 'BoilerNo' point is inserted at the first '%ld' marker and the value of the 'BoilerTemp' point is inserted at the second '%ld' marker and the resulting string is stored in the point 'TextPoint'.

If the value of 'BoilerNo' was 7 and the value of 'BoilerTemp' was 43 then the resultant text stored in the 'TextPoint' is as follows:

```
"Boiler 7 temperature is 43 degrees."
```

Note: The formatting characters are standard 'C' formatting characters (as used by the C-language sprintf function). Some commonly used types are:

- %ld. Insert integer value;
- %f. Insert decimal value. Prefix with decimal point and number to control position (for instance '%.2f' for 2 decimal places);
- %s. Insert string;
- %IX. Insert hexadecimal value (upper case HEX characters, for instance 'FFFF');
- %lx. Insert hexadecimal value (lower case HEX characters, for instance 'ffff');
- %c. Insert character (can be used to convert value to character, for instance to insert control character).

With the text left aligned, and with a width field (for instance '%-6ld' to insert a value left aligned with a field 6 characters wide).

References

More complex expressions (for instance controlling justification, decimal places, number base, etc.) are also possible. Refer to any C language reference book for full details of the format used by the 'sprintf' function.

GetTextLength

Syntax

```
value = GetTextLength (textpoint)
```

Remarks

Argument	Type	Description
textpoint	text	This is the point which has its text length counted.
returnpoint	Integer / real	This is the point that holds the return *value*.

Typical Example

```
textpoint = "Hello World"
count = GetTextLength (textpoint)
```

The number of characters in 'textpoint' is counted and the point 'count' is set to the value 11.

Hex

Syntax

```
result = Hex (value)
```

Remarks

Argument	Type	Description
Value	---	Number to be converted to a Hex number.
Result	---	String containing Hex representation of *value.*

Typical Example

```
HStr = Hex (44)
```

In this example, 'Hstr' contains '2C'.

Left

Syntax

```
lefttext = Left(textpoint,noofchars)
```

Remarks

Argument	Type	Description
textpoint	text	The text point containing the string that is to be manipulated.
noofchars	integer	The number of characters to extract from the start of the string.
lefttext	text	Text point containing the specified range of characters.

Typical Example

```
textpoint = "abcdefgh"
lefttext = Left(textpoint,3)
```

The text point 'lefttext' contains the string 'abc'.

Message

Syntax

```
Message("message")
```

Remarks

Argument	Type	Description
message	string	Contains the text string that is displayed in the message box.

Typical Example

```
Message("this is a message")
```

The message 'this is a message' is displayed in a Message Box.

Mid

Syntax

```
midtext = Mid(textpoint,offset,noofchars)
```

Remarks

Argument	Type	Description
textpoint	text	The text point containing the string that is to be manipulated.
offset	integer	The zero based index of the first character in the string that is to be included in the extract.
noofchars	integer	The number of characters to extract from the string.
midtext	text	Text point containing the specified range of characters.

Typical Example

```
textpoint = "abcdefgh"
midtext = Mid(textpoint,3,2)
```

The text point 'midtext' contains the string 'de'.

PrintMessage

Syntax

```
PrintMessage ("message")
```

Remarks

Argument	Type	Description
message	string	Contains the text string that is sent to the printer.

Typical Example

```
PrintMessage ("Print this message")
```

The message 'print this message' is printed to the configured 'Alarm/message printer', queued if operating in page mode, or printing has been disabled by the EnablePrinting command.

References

Refer to the *CX-Supervisor User Manual* for further details to configure the 'Alarm/message printer'.

Right

Syntax

```
righttext = Right(textpoint,noofchars)
```

Remarks

Argument	Type	Description
textpoint	text	The text point containing the string that is to be manipulated.
noofchars	integer	The number of characters to extract from the end of the string.
righttext	integer	Text point containing the specified range of characters.

Typical Example

```
textpoint = "abcdefgh"
righttext = Right(textpoint,3)
```

The text point 'righttext' contains the string 'fgh'.

TextToValue

Syntax

```
valuepoint = TextToValue(textpoint)
```

Remarks

Argument	Type	Description
textpoint	text	The text point containing the string that is to be converted into a number.
valuepoint	integer	A point containing the value returned after conversion from a string.

Typical Examples

```
textpoint = "10"
valuepoint = TextToValue(textpoint)
```

The value 10 is assigned to the point 'valuepoint'.

```
textpoint = "10.34"
realpoint = TextToValue(textpoint)
```

The real value 10.34 is assigned to the real point 'realpoint'.

ValueToText

Syntax

```
textpoint = ValueToText(value)
```

Remarks

Argument	Type	Description
value	integer	The number that is to be placed into the textpoint. A point name is also a valid parameter.
textpoint	text point	A text point containing the value converted into a string.

Typical Examples

```
textpoint = ValueToText(10)
```

The value 10 is put into a string and assigned to the text point 'textpoint'.

```
value = 10
textpoint = ValueToText(value)
```

This has the same effect as the previous example.

Event/Error Commands

ClearErrorLog

Syntax

```
ClearErrorLog()
```

Typical Example

```
ClearErrorLog()
```

The error list is cleared and the log deleted.

CloseErrorLog

Syntax

```
returnstate = CloseErrorLog()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
CloseErrorLog()
```

The list of all currently logged errors is closed.

DisplayErrorLog

Syntax

```
returnstate = DisplayErrorLog()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
DisplayErrorLog()
```

A list of all currently logged errors is displayed in a dialog.

EnableErrorLogging

Syntax

```
returnstate = EnableErrorLogging(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pointname	bool	A Boolean point.

Typical Example

```
EnableErrorLogging(flag)
```

Error Logging is enabled based on the Boolean point 'flag'. If 'flag' is 'TRUE', then error logging is enabled. If 'flag' is false, then error logging is disabled.

LogError

Syntax

```
returnstate = LogError("message", priority)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
message	string	Contains the text string that is displayed in the Error Log.
priority	integer	Priority assigned to the error. 0 - low 1- medium 2- high.

Typical Example

```
LogError("This is an error", 1)
```

The message 'This is an error' appears as a medium priority error in the error log.

LogEvent

Syntax

```
returnstate = LogEvent("message")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
message	string	Contains the text string that is displayed in the Error Log.

Typical Example

```
LogEvent("this is an event")
```

The message 'this is an event' appears as an event in the error log.

Printer Commands

ClearSpoolQueue

Syntax

```
returnstate = ClearSpoolQueue()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
ClearSpoolQueue()
```

Any messages (typically printed alarms) that are queued up waiting to be sent to the CX-Supervisor Alarm/Message printer is discarded.

EnablePrinting

Syntax

```
returnstate = EnablePrinting(flag)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
flag	bool	0 to disable, 1 to enable.

Typical Example

```
EnablePrinting(FALSE) - Disables printing
EnablePrinting(TRUE) - Enables printing
```

While alarm printing is disabled, any new messages are stored but not printed. When alarm printing is re-enabled, any pending messages are printed (if in line mode) or added to the current page (if in page mode).

GetSpoolCount

Syntax

```
returnstate = GetSpoolCount()
```

Remarks

Argument	Type	Description
returnstate	int	Number of messages queued up waiting to be printed on Alarm/Message printer.

Typical Example

```
NumberMessages = GetSpoolCount()
```

The count of the number of messages (typically printed alarms) that are queued up waiting to be sent to the CX-Supervisor Alarm/Message printer is returned.

PrintActivePage

Syntax

```
returnstate = PrintActivePage(flag)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
flag	bool	Flag is to indicate whether the print setup dialog is to be displayed before printing.

Typical Example

```
PrintActivePage(TRUE)
```

The currently active page is sent to the printer. The flag 'TRUE' indicates that the print dialog is displayed. 'FALSE' causes the print dialog not to be shown.

PrintPage

Syntax

```
returnstate = PrintPage ("pagename", flag)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
pagename	string	The name of the page to be printed.
flag	bool	Flag to indicate whether the print setup dialog is to be displayed before printing.

Typical Example

```
PrintPage("page1", TRUE)
```

The CX-Supervisor page is sent to the printer. The flag 'TRUE' indicates that the print dialog is displayed first to allow for printer configuration. If 'FALSE' was specified instead of 'TRUE' then the print dialog is not shown, the page is just printed.

PrintScreen

Syntax

```
returnstate = PrintScreen(flag)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
flag	bool	Flag to indicate whether the print setup dialog is to be displayed before printing.

Typical Example

```
PrintScreen(FALSE)
```

All CX-Supervisor pages currently on view is printed. The flag 'FALSE' indicates that the print dialog is not displayed. A flag of 'TRUE' causes the print dialog to be shown, allowing the user to configure or choose the printer.

PrintSpoolQueue

Syntax

```
returnstate = PrintspoolQueue()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
PrintSpoolQueue
```

Any message (typically printed alarms) that are queued up waiting to be sent to the CX-Supervisor Alarm/Message printer is printed immediately.

SetPrinterConfig

Syntax

```
returnstate StePrintConfig(Driver, Device, Port)
```

Remarks

Argument	Type	Description
returnstate	Bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
Driver	String	Name of printer device (e.g. "Epson9" for 9 pin Epson printers.
Device	String	Name of specific device (e.g. "Epson FX-870"). This is optional.
Port	String	Name of port or file(e.g. "LPT1.").
Line Terminator	String	Optional. Sets terminator (e.g. cr) to be added to end of each printed line.

Typical Examples

```
SetPrinterConfig("SCSPRN", "", "LPT1:")
```

This uses standard CX-Supervisor line print driver.

```
SetPrinterConfig("", "", "")
```

This uses default Windows printer driver.

```
SetPrinterConfig("Epson9", "", "LPT2:")
```

This uses Epson printer driver, attached to LPT2.

```
SetPrinterConfig(DriverNamePoint, DeviceNamePoint, PrintNamePoint)
```

This uses text points.

```
Terminator = FormatText("%c%c",13,10)
```

Character 10 is 'lf' (newline), character 13 is cr (carriage return).

```
SetPrinterConfig("Epson9","","LPT1:",Terminator)
```

Security Commands

Login

Syntax

```
returnstate = Login(username, password)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
username	Text	Optional parameter with name of user to login. If omitted, the login dialog will be shown.
password	Text	Optional parameter with password for user to login. If used, username must be specified, even if only empty i.e. "". If omitted, the login dialog will be shown.

Typical Examples

```
Login()
```

The Login dialog is displayed for user entry.

```
Login("Designer", "Designer")
```

The default 'Designer' user is logged in automatically using matching password.

References

Refer to the *CX-Supervisor User Manual* for details of Login.

Logout

Syntax

```
returnstate = Logout()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
Logout()
```

The user is logged out.

References

Refer to the *CX-Supervisor User Manual* for details of Logout.

Setup Users

Syntax

```
returnstate = SetupUsers()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
SetupUsers()
```

The Setup Users dialog is displayed for user entry.

References

Refer to the *CX-Supervisor User Manual* for details of setting and modifying user details.

Data Logging Commands

ClearLogFile

Syntax

```
ClearLogFile("datasetname")
```

Remarks

Argument	Type	Description
datasetname	string	Name of Data Set to clear as text point or constant.

Typical Example

```
ClearLogFile("Process 1")
```

This command will clear all data from the active (latest) log file for this data set, and add a 'Clear Event' indicator.

CloseLogFile

Syntax

```
returnstate = CloseLogFile("datasetname")
```

or

```
returnstate = CloseLogFile("databaselink")
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0
datasetname	text	Name of Data Set to close as text point or constant.
databaselink	text	Name of Database link to close as text point or constant.

Typical Example

```
CloseLogFile("Process 1")
```

This command will close the active log file for the data set. Logging for this data set is automatically stopped.

CloseLogView

Syntax

```
CloseLogView("datasetname")
```

Remarks

Argument	Type	Description
datasetname	string	Name of Data Set view to close as text point or constant.

Typical Example

```
CloseLogView("Process 1")
```

This command will close the Data Log Viewer, which is displaying the named data set.

ExportAndViewLog

Syntax

```
ExportAndViewLog ("datasetname", "item list", "format", file,
outputfile)
```

or

```
ExportAndViewLog ("datasetname", TextArray, "format", file,
outputfile)
```

Remarks

Argument	Type	Description
datasetname	string	Name of Data Set to export as text point or constant.
item list	string	List of Items and/or Groups within the data set to export, separated by commas. Alternatively use "*" to export all.
TextArray	string array	A text point, which has an array size specified as 1 or more elements . Each element holds an Item or Group name.
format	string	Either "CSV" or "Text" to specify output format. May include suffix '-' followed by: B to exclude break information D to exclude the log date T to exclude the log time M to exclude to log milliseconds
file	integer	Number of file to export where 0 is the latest (active) file, 1 is the previous file etc.
outputfile	string	File name for output file. May include full path, which will be created automatically if it does not exist.

All these arguments are optional, and may be omitted provided there are no further arguments i.e. to specify the 'format', 'datasetname' and 'item list' must be included but 'file' and 'output' may be omitted.

Typical Examples

```
ExportAndViewLog("Balloon", "*")
```

or

```
ExportAndViewLog("Balloon", "Altitude,Fuel,Burning,Lift,Group 1",
"CSV-BDTM", 0, "output")
```

or

```
ItemList[0] = "Altitude"
ItemList[1] = "Fuel"
ItemList[2] = "Burning"
ItemList[3] = "List"
ItemList[4] = "Group 1"

ExportAndViewLog("Balloon", ItemList, "CSV-BDTM", 0, "output")
```

All these commands will export all the data in the specified file, for the named data set to the named output file, in the format specified (as per ExportLog). It then launches an appropriate viewer to display the file, using the Windows file associations.

ExportLog

Syntax

```
ExportLog ("datasetname", "item list", "format", file, outputfile)
```

or

```
ExportLog ("datasetname", TextArray, "format", file, outputfile)
```

Remarks

Argument	Type	Description
datasetname	string	Name of Data Set to export as text point or constant.
item list	string	List of Items and /or Groups within the data set to export, separated by commas. Alternatively use "*" to export all.
TextArray	string array	A text point, which has an array size specified as 1 or more elements. Each element holds an Item or Group name.
format	string	Either "CSV" or "Text" to specify output format. May include suffix '-' followed by: B to exclude break information D to exclude the log date T to exclude the log time M to exclude to log milliseconds
file	integer	Number of file to export where 0 is the latest (active) file, 1 is the previous file etc.
outputfile	string	File name for output file. May include full path, which will be created automatically if it does not exist.

All these arguments are optional, and may be omitted provided there are no further arguments i.e. to specify the 'format', 'datasetname' and 'item list' must be included but 'file' and 'output' may be omitted.

Typical Examples

```
ExportLog("Balloon", "*")
```

or

```
ExportLog("Balloon", "Altitude,Fuel,Burning,Lift,Group 1" "CSV-BDTM",
0, "output")
```

or

```
ItemList[0] = "Altitude"
ItemList[1] = "Fuel"
ItemList[2] = "Burning"
ItemList[3] = "List"
ItemList[4] = "Group 1"
```

```
ExportAndViewLog("Balloon", ItemList, "CSV-BDTM", 0, "output")
```

All these commands will export all the data in the specified file, for the named data set to the named output file, in the format specified.

OpenLogFile

Syntax

```
returnstate = OpenLogFile("datasetname")
```

or

```
returnstate = OpenLogFile("databaselink")
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0
datasetname	text	Name of Data Set to open as text point or constant.
databaselink	text	Name of Database link to open as text point or constant.

Typical Example

```
OpenLogFile("Balloon")
```

This command will open the log file, ready to start logging. As the function is disk intensive it should not be called frequently.

OpenLogView

Syntax

```
OpenLogView("datasetname", "item list", sessionfile)
```

or

```
OpenLogView("datasetname", TextArray, sessionfile)
```

Remarks

Argument	Type	Description
datasetname	string	Name of Data Set to view as text point or constant.
item list	string	List of Items and/or Groups within the data set to view, separated by commas
TextArray	string array	A text point, which has an array size specified as 1 or more elements. Each element holds an Item or Group name.
sessionfile	string	Optional filename of session information file. The Data Log Viewer is shown with the session settings (e.g. Window position, size, colours, grid options etc. stored in the session file. If omitted, the previous settings are used.

Typical Example

```
OpenLogView("Balloon", "Altitude,Fuel,Burning,Lift,Group 1")
```

or

```
ItemList [0] = "Altitude"
ItemList [1] = "Fuel"
ItemList [2] = "Burning"
ItemList [3] = "Lift"
ItemList [4] = "Group 1"
OpenLogView("Balloon", ItemList)
```

Both these commands will open the Data Log Viewer, and load the Balloon log file, and show the named items.

```
OpenLogView("Balloon", ItemList, "C:\Program Files\Omron\CX-
SUPERVISOR\App\MySessionInfo.txt")
```

This command will open the Data Log Viewer and Balloon log file as above but the Data Log Viewer will always appear in the same position, and with the same settings – not as it was last shown.

StartLogging

Syntax

```
returnstate = StartLogging("datasetname")
```

or

```
returnstate = StartLogging("databaselink")
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0
datasetname	text	Name of Data Set to start logging as text point or constant.
databaselink	text	Name of Database link to start logging as text point or constant.

Typical Example

```
StartLogging("Process 1")
```

This command will start logging of all items in the named data set. If the file is closed it will be automatically opened.

StopLogging

Syntax

```
returnstate = StopLogging("datasetname")
```

or

```
returnstate = StopLogging("databaselink")
```

Remarks

Argument	Type	Description
returnstate	bool	Optional. 1 if the function is successful otherwise 0
datasetname	Text	Name of Data Set to stop logging as text point or constant.
databaselink	text	Name of Database link to stop logging as text point or constant.

Typical Example

```
StopLogging("Process 1")
```

This command will stop logging of all items in the named data set.

Database Commands

DBAddNew

Description

Adds a new record to a Recordset. This function will fail if the Recordset is opened with a lock of 'Read Only'.

Syntax

```
returnstate = DBAddNew(level)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This should be a field or recordset level.

Typical Examples

```
Result = DBAddNew("Northwind.Order Details")
```

Using a Recordset connection level, a new record is added with values from all fields associated with a property type 'Add'. Point 'Result' is set true if this was successful.

```
DBAddNew("Northwind.Order Details.OrderID")

DBAddNew("Northwind.Order Details.ProductID")
DBAddNew("Northwind.Order Details.Quantity")

DBAddNew("Northwind.Order Details.UnitPrice")

DBUpdate("Northwind.Order Details")
```

Using a Field connection level, each required field is added to the new record using multiple calls to DBAddNew(). When the record is complete, it is added by calling the DBUpdate() function

Note: To use DBAddNew() with a Recordset level the Recordset must be configured to perform this type of operation i.e. it will need to contain fields for any primary keys and 'non null' values required to create a new record. When used at Recordset level all fields associated with the Recordset with property type 'Add' are added (as if calling DBAddNew()) and the record is updated (as if calling DBUpdate()). Points associated with the 'Add' property can be array points, thus enabling you to add multiple records in one operation.

Note: When using a Field level connection, the operation may be cancelled at any stage before the DBUpdate() function is called by calling the DBExecute() command "CancelUpdate".

Note: Only Fields with a property type of 'Add' can be added to a Recordset. The value(s) of the associated points at the time DBUpdate() is called will be used to create the record.

DBClose

Description

Closes a Connection or Recordset. Closing a Connection will automatically close all recordsets associated with it. Recordsets can be closed in isolation by selecting the appropriate level.

Syntax

```
returnstate = DBClose(level)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This should be a connection or recordset level.

Typical Examples

```
Result = DBClose("Northwind.Order Details")
```

Closes the 'Order Details' Recordset

```
Result = DBClose("Northwind")
```

Closes the connection to the Northwind database, and also any Recordsets which may be open.

DBDelete

Description

Deletes the specified number of records from the current record position. This function works only at the Recordset level. This function will fail if the Recordset is opened with a lock of 'Read Only'.

Syntax

```
returnstate = DBDelete(level, quantity)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This must be a recordset level.
quantity	int	Number of records to delete.

Typical Examples

```
Result = DBDelete("Northwind.Order Details", 10)
```

Delete the next 10 records in the recordset

```
DBMove("First")
Result = DBDelete("Northwind.Order Details", 10)
```

Delete the first 10 records.

DBExecute

Description

The DBExecute function allows the execution of miscellaneous commands and allows for future expansion by supporting new commands without the need to create more new DB functions.

Syntax

```
return = DBExecute(level, command, parameter)
```

Remarks

Argument	Type	Description
return		1 if the function is successful otherwise 0 except for "Find" and "FindNext" commands which return the record number if found or if not, set the current record to EOF and return -1.
level	text	A text point or constant specifying the connection level, which depends on the command specified.
command	text	Command to execute. May be one of the commands listed below.
parameter	text	Command parameter only required with certain commands. For "Connection", this parameter should hold the new connection string. For "Find" and "FindNext" this parameter should be the search criteria. For "Source" this is the Recordset source. For "Filter" this is the Recordset filter.

Typical Examples

```
Pos = DBExecute("Northwind.Order Details", "Find", "UnitPrice >
14.00")
```

Find the next record satisfying the specified criteria, starting from the current position. Valid search criteria include: "ProductName LIKE 'G*' " wildcard search finds all records where ProductName starts with 'G', "Quantity = 5", "Price >= 6.99". Only single search values are allowed, using multiple values with 'AND' or 'OR' will fail.

```
DBExecute("Connection1.Recordset1", "Source", "Table2")
```

Modify the Recordsets source to open a different table than configured.

```
DBExecute("Northwind.Shippers", "Filter", "CompanyName = 'United
Package'")
```

Apply a filter to display only records with a company name 'United Package'

```
DBExecute("Northwind.Shippers", "Filter", "")
```

Cancel an existing filter (by passing an empty string)

DBExecute Commands

Command	Connection Level	Description
Connection	Connection	Modify the connection string.
BeginTrans	Connection	Begins a new Transaction.
CommitTrans	Connection	Saves any pending changes and ends the current transaction.
RollbackTrans	Connection	Cancels any changes made and ends the transaction.
CommitTransAll	Connection	Saves all changes and ends all transactions.
RollbackTransAll	Connection	Cancels all changes and ends all transactions.
TransCount	Connection	Returns the number of pending transactions.
Requery	Recordset	Re-run the Recordset Query.
CancelUpdate	Recordset	Cancel a DBAddNew operation.
Find	Recordset	Find the specified criteria in a Recordset.
FinNext	Recordset	Combined DBMove("Next"), DBFind() operation.
Source	Recordset	Modify the Recordset source.
Filter	Recordset	Apply a filter to a Recordset.
Save	Recordset	Saves a Recordset in XML format.

DBGetLastError

Description

Returns the last error string generated by the Database provider, and displays it in a message box.

Syntax

```
returnstate = DBGetLastError(level, display)
```

Remarks

Argument	Type	Description
returnstate	text	The error message from the provider
level	text	A text point or constant specifying the connection level. This must be a Connection level.
display	bool	Optional flag. By default DBGetLastError will display the providers error message in a message box. Setting this flag to FALSE prevents this action.

Typical Examples

```
DBGetLastError("Northwind")
```

or

```
DBGetLastError("Northwind", TRUE)
```

Both the above lines will get and display the last error to occur for the Northwind connection.

```
ErrMsg = DBGetLastError("Northwind", FALSE)
```

The last error to occur for the Northwind connection is stored Text point 'ErrMsg', without displaying a message box.

DBMove

Description

The DBMove function enables you to navigate around a Recordset by moving the position of the 'current record' in the Recordset. When a Recordset is first opened the first record is the current record.

Syntax

```
returnstate = DBMove(level, direction, position)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This must be a Recordset level.
direction	text	A text string indicating where to move to. May be one of: "First" "Last" "Next" "Previous" "Position" "FirstPage" "LastPage" "NextPage" "PreviousPage" "Page" "Bookmark"
position	int/real	This optional parameter is only required when directions of "Position", "Page" and "Bookmark" are used. When used with "Position" and "Page" this parameter must be an integer, and is the record or page number to move to. When used with "Bookmark" this parameter must be a real.

Typical Examples

```
DBMove("Northwind.Order Details", "First")
```

Go to the first record in the Recordset.

```
pos = 3
DBMove("Northwind.Order Details", "Position", pos)
```

Go to the third record in the Recordset.

```
DBMove("Northwind.Order Details", "Page", 6)
```

Go to the sixth page in the Recordset.

Note: Bookmarks are returned from the function 'DBProperty', they enable you to return to a 'marked' record, even after records have been added or deleted

Note: Some Providers do not support moving in the "Previous" direction i.e. cursors are 'Forward-Only'. Some 'Forward-Only' providers do allow moving "First", while some are strictly Forward-Only i.e. the Recordset has to be Re-queried effectively a combined Close then Open operation to reset the cursor back to the start of the Recordset. Some Providers that do support moving "Previous" do not support moving to "Position". However, in order to be consistent, CX-Supervisor ensures that that all operations (except "Bookmarks") will work for any connection to any provider but you need to bear in mind when designing applications that use 'Forward-Only' cursors, that there may be some 'long-winded' acrobatics being performed behind the scenes. See DBSupports() for details of how to check the type of cursor in force.

Note: Bookmarks will only work if specifically supported by the Provider.

DBOpen

Description

Opens a Connection or Recordset. Opening a Connection will automatically open all recordsets associated with it, that are marked as auto open. Recordsets can be opened in isolation by selecting the appropriate level.

Syntax

```
returnstate = DBOpen(level)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This may be a Connection or Recordset level.

Typical Examples

```
DBOpen("Northwind")
```

Open the connection to the Northwind database, and automatically open any Recordsets set to open on connection.

```
done = DBOpen("Northwind.Order Details")
```

Just open a specific Recordset.

DBProperty

Description

Returns the requested property. This function operates on the Recordset and Field levels. The type of the value returned depends on the property requested.

Syntax

```
returnstate = DBProperty(level, property)
```

Remarks

Argument	Type	Description
returnstate		Property value returned. See table for type.
level	text	A text point or constant specifying the connection level. This may be a Recordset or Field level.
property	text	The name of the property to get. For details see the Recordset Properties and Field Properties tables.

Typical Examples

```
Page = DBProperty("CSV.Result", "CurrentPage")
```

Get the current page for the CSV.Result Recordset.

```
FieldSize = DBProperty("Northwind.Customers.Address", "Size")
```

Get the size for the 'Address' field.

Note: The Recordset will only return valid properties when it is Open.

Recordset Properties

The properties of a Recordset are:

Property	Description	Return type
"CurrentRecord"	Current cursor position	Integer
"RecordCount"	Number of records in the Recordset.	Integer
"Bookmark"	Record marker.	Real
"PageCount"	Number of pages in the Recordset.	Integer
"PageSize"	Number of records in a page.	Integer
"CurrentPage"	Page in which the cursor position resides.	Integer
"Source"	Command or SQL that created the Recordset.	Text
"Sort"	Field name(s) the Recordset is sorted on.	Text
"FieldCount"	Number of fields(columns) in the Recordset.	Integer
"BOF"	Current position is at the start of the Recordset.	Bool
"EOF"	Current position is at the end of the Recordset.	Bool

Field Properties

The properties of a Field are

Property	Description	Return type
"Value"	Value of the field at the current position.	As type of field
"Name"	Name of the Field.	String
"Type"	The fields data type.	String
"Size"	Maximum width of the field.	Integer

DBRead

Description

Reads a record from a Recordset to the associated point(s), or if associated points are array points, reads a whole page of records. This function operates on both Recordset and Field levels. At the Field level the associated column values from the Recordsets current position will be copied into the Point (number of elements copied = number of elements in the Point, no paging applies at the Field level).

Syntax

```
returnstate = DBRead(level, reset)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This may be a Recordset or Field level.
reset	bool	This argument is optional and may be omitted. If omitted or TRUE, when the read is complete the record cursor is reset to the position prior to reading.

Typical Examples

```
DBRead("Northwind.Customers")
```

Read the next page of records from the 'Customers' Recordset.

```
DBRead("Northwind.Customers", FALSE)
```

Read the next page of records from the 'Customers' Recordset, and leave the cursor at the next record.

```
DBRead("Northwind.Customers.Address")
```

The Address field is read. If it is an array point, the Address is read from subsequent records until the array has been filled.

Note: Use with reset = TRUE is useful if the read operation is being combined with a subsequent Write operation i.e. you can read in a set of records - resetting the cursor, make modifications to some of the fields and then Write the changes back to the Recordset.

Note: Use with reset = FALSE will leave the current position at the start of the next set of records. This option can be of benefit if the Provider only supports forward moving cursors, or you simply want to step through the records a page at a time.

DBSchema

Description

Issues commands to read schema results or properties or set up new schema criteria. This function operates only at a Schema level.

Syntax

```
return = DBSchema(level, command, parameters...)
```

Remarks

Argument	Type	Description
return		Value returned by command. For some commands e.g. "RecordCount" this is an integer value, for other commands this is a text value.
level	text	A text point or constant specifying the connection level. This must be a Schema level.
command	text	The command must be one of the following: "Read" - Transfers a schema page into the associated point "Set" - Enables schema details to be modified "Type" - Returns the current Schema Type "Criteria" - Returns the current Schema Criteria "Filter" - Returns the current Schema Filter "RecordCount" - Returns the number of records in the current Schema "PageCount" - Returns the number of pages in the current Schema "CurrentPage" - Returns the current Schema page
parameters		Some commands require 1 or more extra parameters. "Read" takes an optional parameter 'Page Number' of type integer. If no 'Page Number' is supplied, this function will return page 1 when first called and automatically return the next page of schemas for each subsequent call, cycling back to the beginning when all pages have been returned. "Set" takes three text parameters for Schema 'Name', 'Criteria' and 'Filter'.

Typical Examples

```
NumberOfRecords = DBSchema("Invoice.Data Types", "RecordCount")
```

Read the Number of records in the Schema.

```
DBSchema("Invoice.Data types", "Read", 2)
```

Read Schema page 2 results into the associated point.

```
DBSchema("Invoice.Data Types", "Set", "Columns", "COLUMN_NAME", "")
```

Set a new Schema to return column names.

DBState

Description

Reports if the specified level is in the requested state.

Syntax

```
return = DBState(level, state)
```

Remarks

Argument	Type	Description
return	bool	1 if the specified level is in the requested state, otherwise 0
level	text	A text point or constant specifying the connection level. This may be a Connection or Recordset level.
state	text	The requested state must be either "Open" or "Closed"

Typical Examples

```
State = DBState("Invoice", "Closed")
```

Checks if the Connection "Invoice" is currently closed.

```
State = DBState("Northwind.Customers", "Open")
```

Checks if the Recordset "Customers" is currently open.

DBSupports

Description

Returns TRUE if the specified Recordset supports the requested operation.

Syntax

```
return = DBSupports(level, operation)
```

Remarks

Argument	Type	Description
return	bool	1 if the specified Recordset supports the requested operation, otherwise 0
level	text	A text point or constant specifying the connection level. This must be a Recordset level.
operation	text	The requested operation may be one of: "AddNew", "Bookmark", "Delete", "Find","MovePrevious" or "Update"

Typical Example

```
Result = DBSupports("CSV.Recordset1", "Delete")
```

Checks if records can be deleted in 'Recordset1'

Note: If the "MovePrevious" operation is not supported then only 'Forward-Only' cursor movements are supported.

DBUpdate

Description

Update the record being added in a Recordset. Used in conjunction with DBAddNew to commit a new record.

Syntax

```
returnstate = DBUpdate(level)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0
level	text	A text point or constant specifying the connection level. This must be a Recordset level.

Typical Example

```
DBAddNew("Northwind.Order Details.OrderID")
DBAddNew("Northwind.Order Details.ProductID")
DBAddNew("Northwind.Order Details.Quantity")
DBAddNew("Northwind.Order Details.UnitPrice")
DBUpdate("Northwind.Order Details")
```

Each field is added to the new record using multiple calls to DBAddNew(). When the record is complete, it is added to the Recordset by calling the DBUpdate() function.

DBWrite

Description

Writes a set of records into a Recordset from the associated point(s). This function operates on both Recordset and Field levels. At the Recordset level all the associated points values from the Points will be written into the Recordset starting at the current record (1 page of values will be written for each Point). At the Field level the associated values from the point are written into the Recordsets starting at the current position. The number of elements written = number of elements in the Point. This function will fail, if the Recordset is opened with a Lock of 'Read Only'.

Syntax

```
return = DBWrite(level, reset)
```

Remarks

Argument	Type	Description
return	Bool	1 if the function is successful otherwise 0
level	Text	A text point or constant specifying the connection level. This must be a Recordset level.
reset	Bool	This argument is optional and may be omitted. If omitted or TRUE, when the write is complete the record cursor is reset to the position prior to writing.

Typical Examples

```
DBWrite("Northwind.Customers")
```

Write all point values to the associated Customers fields.

```
DBWrite("Northwind.Customers.Address", FALSE)
```

Write the point values to the Address column, and leave the cursor at the next set of records.

Serial Port Functions

InputCOMPort

Description

Sets the serial communications port for receiving ASCII text messages. Any message received is placed in the text point. The boolean flag is set true to indicate that a message has been received. It is up to the user to reset this flag between receiving messages in order to indicate that a new message is present.

Syntax

```
ReturnState = InputCOMPort(PortNumber, Message, MessagePresent)
```

Remarks

Argument	Type	Description
ReturnState	Bool	True if successful else false.
PortNumber	Integer	The number of the port previously configured using the function SetupCOMPort and opened with OpenCOMPort.
message	Text	Text point to hold ASCII text message received through the port.
MessagePresent	Bool	Boolean point indicating that a message has been received.

Typical Example:

```
bState = InputCOMPort(1, Msg, bTransmission)
```

OutputCOMPort

Description

Sends an ASCII text message out through the designated serial communications port.

Syntax

```
ReturnState = OutputCOMPort(PortNumber, Message)
```

Remarks

Argument	Type	Description
ReturnState	Bool	True if successful else false.
PortNumber	Integer	The number of the port previously configured using the function SetupCOMPort and opened with OpenCOMPort.
message	Text	Text point holding the ASCII text message to send through the port.

Typical Example:

```
bState = OutputCOMPort(1, Msg)
```

CloseCOMPort

Description

Closes the designated serial communications port on the PC. The port must have been configured and opened before it can be closed.

Syntax

```
ReturnState = CloseCOMPort(PortNumber)
```

Remarks

Argument	Type	Description
ReturnState	Bool	True if successful else false.
PortNumber	Integer	The number of the port previously configured using the function SetupCOMPort and opened using the script function OpenCOMPort.

Typical Example:

```
bState = CloseCOMPort(1)
```

OpenCOMPort

Description

Opens the designated serial communications port on the PC for transmitting or receiving data. The port must have been configured before it can be opened.

Syntax

```
ReturnState = OpenCOMPort(PortNumber)
```

Remarks

Argument	Type	Description
ReturnState	Bool	True if successful else false.
portNumber	Integer	The number of the port previously configured using the function SetupCOMPort.

Typical Example:

```
bState  = OpenCOMPort(1)
```

SetupCOMPort

Description

Configures the designated serial communications port on the PC for transmitting or receiving data.

Syntax

```
ReturnState = SetupCOMPort(PortNumber, ConfigurationString,
HandShaking, TerminationChar, ControlCharFlag)
```

Remarks

Argument	Type	Description
returnstate	Bool	True if successful else false.
portnumber	Integer	The number of the serial port to be configured.
ConfigurationString	Text	A string indicating the desired Baud rate, Parity, number of data bits and stop bits.
HandShaking	Integer	The required handshaking protocol. Valid values are 0 – None 1 - XonXoff 2 – RTS 3 - RTS & XonXoff
TerminationChar	Integer	A character indicating the end of the message.
ControlCharFlag	Bool	A flag indicating that control characters contained in a received message should be Ignored.

Typical Example:

```
bState  = SetupCOMPort(2, "9600,N,8,1", 0, 0x0D, TRUE)
```

ActiveX Functions

GetProperty

Description

Gets the value of a property of an OLE object and stores it in a point.

Syntax

```
propertyvalue = GetProperty(object, property, ...)
```

Remarks

Argument	Type	Description
propertyvalue	n/a	The value of the property. Type is dependant on the type of the property.
object	Text	The name of the OLE object to get the property of.
property	Text	The name of the property to get.
. . .	n/a	Any number of parameters for the property.

Typical Examples

```
OLE1Height = GetProperty("OLE1", "Height")
```

This will read the property 'Height' from the OLE object 'OLE1' and store it in the point 'OLEHeight'.

```
DM100Value = GetProperty("CXComms1", "DM", 100)
```

This will read the property 'DM' (with one parameter 100) from the OLE object 'CXComms1' and store it in the point 'DM100Value'.

PutProperty

Description

Puts a value stored in a point into the property of an OLE object.

Syntax

```
PutProperty(object, property, ..., value)
```

Remarks

Argument	Type	Description
object	Text	The name of the OLE object containing the property to change.
property	Text	The name of the property to put.
. . .	n/a	Any number of parameters for the property.
value	n/a	The value to write to the property. Type is dependant on the type of property. Can also be a number.

Typical Examples

```
PutProperty("OLE1", "Left", NewLeftValue)
```

This will write the value stored in the point NewLeftValue to the property 'Left' in the OLE object 'OLE1'.

```
PutProperty("CXComms1", "DM" 10, NewValue)
```

This will write the value stored in the point NewValue to the property 'DM' (with one parameter 10) in the OLE object 'CXComms1'.

```
PutProperty("Gauge1", "Value", 25.2)
```

This will write the value 25.2 to the object 'Gauge1'.

Execute

Description

Execute a method of an OLE object.

Syntax

```
Execute(object, method, ...)
```

Remarks

Argument	Type	Description
object	Text	The name of the OLE object.
method	Text	The name of the method to execute.
. . . .	n/a	Any number of parameters for the method.

Typical Examples

```
Execute("OLE1", "Start")
```

This will call the method 'Start' on the object 'OLE1'.

```
Execute("CXComms1", "OpenPLC", "MyPLC")
```

This will call the method 'OpenPLC' with one text parameter 'MyPLC' on the OLE object 'CXComms1'

ExecuteVBScript

Description

Creates aliases allowing Visual Basic Script to be executed in line. This uses the Windows Scripting Host. See chapter 5 for a list of supported functions and details of the Windows Scripting Host.

Syntax

```
@VBSCRIPT
@ENDSCRIPT
```

Typical Examples

```
@VBSCRIPT
     OLE1.LEFT = Point("PointName")
@ENDSCRIPT
```

This Visual Basic Script will write the value from the point 'PointName' into the property 'Left' of the OLE object 'OLE1'.

ExecuteJScript

Description

Creates aliases allowing Java Script to be executed in line. See Appendix C for a list of supported functions and details of the Windows Scripting Host.

Syntax

```
@JSCRIPT
@ENDSCRIPT
```

Typical Examples

```
@JSCRIPT
    Point("PointName") = OLE_1.Height;
@ENDSCRIPT
```

This Java Script will write the value of the property 'Height' from the OLE object 'OLE1' into the Point named 'PointName'.

Note: The Java Script can not include the { or } characters. To use these, put the script in a text file and use the ExecuteJScriptFile function.

ExecuteVBScriptFile

Description

Allows Visual Basic script stored in a text file to be executed. This uses the windows scripting host which must be installed. See chapter 5 for a list of supported functions.

Syntax

```
returnstate = ExecuteVBScriptFile(scriptfile)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
scriptfile	Text	The name of the file with the Visual Basic Script to execute.

Typical Examples

```
returnstate = ExecuteVBScriptFile("c:\vbscript.txt")
```

This will execute the Visual Basic Script stored in "c:\vbscript.txt".

ExecuteJScriptFile

Description

Allows Java script stored in a text file to be executed. This uses the windows scripting host which must be installed. See Appendix C for a list of supported functions.

Syntax

```
returnstate = ExecuteJScriptFile(scriptfile)
```

Remarks

Argument	Type	Description
returnstate	bool	1 if the function is successful otherwise 0.
scriptfile	Text	The name of the file with the Java Script to execute.

Typical Examples

```
returnstate = ExecuteJScriptFile("c:\jscript.txt")
```

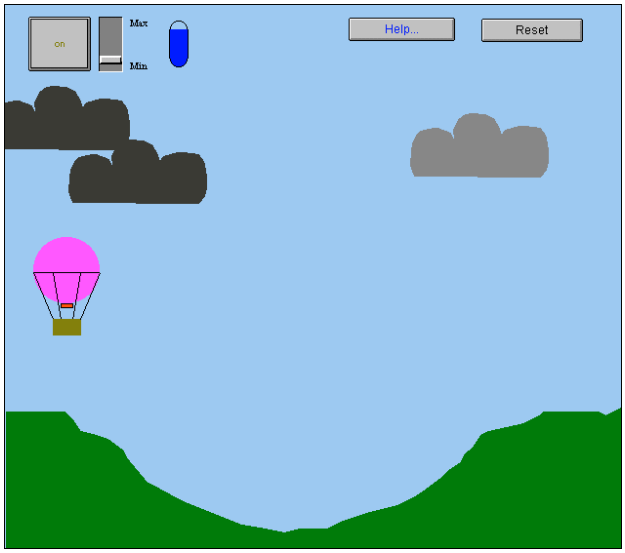
This will execute the Java Script stored in "c:\jscript.txt".

CHAPTER 7
Script Example

This chapter provides an example application for a script. The script is a typical script exercising the basic commands. It is described twice, once as a whole, and once on a line by line basis.

Balloon Script

The following script applies to a simple game.



The user must attempt to land the balloon on the plateau on the right, using the ***Max/Min*** slider control throughout the flight. Clicking ***Reset*** clears the current game and initialises a new game. Clicking the ***on/off*** pushbutton starts the game.

When the balloon is airborne, clouds move slowly horizontally and change colour slightly. Clicking ***Help*** at any time brings up a special help page; clicking ***Close*** from this help page returns the user to the game. The blue gauge shows the amount of fuel consumed and left.

The project consists of three page scripts and one object. The three page scripts are initiated at varied intervals: 10 milliseconds, 100 milliseconds and 1000 milliseconds.

The page script initiated at intervals of 10 milliseconds determines the position of each cloud, and the speed at which each cloud moves. The page script initiated at intervals of 1000 milliseconds determines how the balloon reacts to the conditions.

The page script initiated at intervals of 100 milliseconds provides the main configuration of the game, reacting to user input and moving the balloon accordingly. This page script is as follows:

```
IF burner AND alt > 400.0 THEN
  burner = FALSE
ENDIF
IF burner THEN
  fuel = fuel - rate
  IF fuel < 0.0 THEN
    fuel = 0.0
    burner = FALSE
  ENDIF
ENDIF

IF burner AND fuel > 0.0 AND rate > 0.0 THEN
  lift = lift + rate/5.0
ELSE
  IF alt > 140.0 THEN
    lift = lift - 0.2
  ENDIF
ENDIF
IF lift < -10.0 THEN
  lift = -10.0
ENDIF
alt = alt + lift
IF alt <= 140.0 THEN
  IF distance>630.0 AND distance<660.0 AND lift>=-3.0 THEN
    winner = TRUE
    burner = FALSE
  ENDIF
  IF lift < -3.0 then
    crash = TRUE
    burner = FALSE
  ENDIF
  lift = 0.0
ENDIF

speed = (alt-140.0 )/100.0
IF speed < 0.0 then
  speed = 0.0
ENDIF

distance = distance + speed
```

The following paragraphs describe the above script on a line by line basis.

```
IF burner AND alt > 400.0 THEN
  burner = FALSE
ENDIF
```

If the fuel burner is on, based on Boolean point 'burner' set to 'TRUE', and the altitude of the balloon, based on point 'alt', exceeds 400, then the fuel burner is turned off. Point 'alt' is measured in pixels between 140 and 1000, so the value of 400 is the height in pixels.

```
IF burner THEN
  fuel = fuel – rate
  IF fuel < 0.0 THEN
    fuel = 0.0
    burner = FALSE
  ENDIF
ENDIF
```

If the fuel burner is on, the amount of fuel remaining decreases by the rate of ascent. The rate of ascent, point 'rate' can be modified by moving the slider. If point 'fuel' currently has a value of less than 0, then there is no fuel left and the fuel burner is turned off.

```
IF burner AND fuel > 0.0 AND rate > 0.0 THEN
  lift = lift + rate/5.0
ELSE
  IF alt > 140.0 THEN
    lift = lift - 0.2
  ENDIF
ENDIF
```

If the fuel burner is on, and there is still fuel left, and the rate of ascent exceeds 0 (the balloon has taken off) then point 'lift' is incremented by the rate of ascent divided by 5 to allow the balloon to climb. Otherwise the balloon must be descending and point 'lift' is decremented by 0.2.

```
IF lift < -10.0 THEN
  lift = -10.0
ENDIF
```

Once point 'lift' reaches -10, it is not allowed to go lower.

```
alt = alt + lift
```

The altitude of the balloon is incremented by point 'lift'.

```
IF alt <= 140.0 THEN
  IF distance>630.0 AND distance<660.0 AND lift>=-3.0 THEN
    winner = TRUE
    burner = FALSE
  ENDIF
```

If the balloon has hit the ground (point 'alt' equals 140), then provided it is on the plateaux (the position of the balloon in pixels defined by point 'distance' is between 630 and 660) and the rate of descent is not too fast (defined by point 'lift'), then the game is won.

```
  IF lift < -3.0 then
    crash = TRUE
    burner = FALSE
  ENDIF
```

If the balloon has hit the ground (point 'alt' equals 140), then if the rate of descent is not too fast (defined by point 'lift'), then the game is lost.

```
    lift = 0.0
ENDIF
```

Point 'lift' is reset.

```
speed = (alt-140.0 )/100.0
IF speed < 0.0 then
  speed = 0.0
ENDIF
```

Point 'speed' is calculated based on the altitude.

```
distance = distance + speed
```

Point 'distance' is calculated based on the speed.

CHAPTER 8
Colour Palette

This chapter discusses the colour palette. A colour may be specified by its name or number. The following table provides a cross-reference between these. Some colour names made up of more than one word are separated by an underscore or a hyphen. A specified colour can be changed in the CX-Supervisor development environment for the current session; such changes cannot be saved to a Page or Project, unless colours are changed from the Colour Palette located under the *General Settings* submenu in the *Project* menu.

Using a 16 colour-based screen resolution (consult the Microsoft Windows documentation for further information) colours 16 to 65 are dithered from the sixteen base colours. Higher colour-based resolutions are not dithered.

No.	Colour	No.	Colour
0	black	12	purple
1	blue	13	olive
2	green	14	dark_grey
3	cyan	15	light-grey
4	red	16	pale-green
5	magenta	17	light-blue
6	yellow	18	off-white
7	white	19	grey
8	dark_blue	20	cherry
9	dark_green	21	silver
10	blue-green	22	apple
11	brown	23	orange
		24-65	Not used

Appendix A
OPC Communications Control

This appendix contains a list of the available component properties and gives details of the Visual Basic script interface. These properties can be set in run time by using a Visual Basic script command – for example: -

```
OMRONCXOPCCommunicationsControl1.ServerNodeName = "\\NAME"
```

The Script Interface defines the Visual Basic script interface for the OPC communications control. See ExecuteVBScript script functions for more information on running Visual Basic Script.

Component Properties

Property Title	Example	Description
DisplayErrors	True False	When set True, the object will display a message box for any errors. If set to False, error messages are not displayed.
ProjectName		Name of .OPC file containing the client setup.
ServerComputerName	"MyPC"	This is the name of the PC with the OPC Server.
ServerName		Name of the OPC Server to connect to. e.g. OMRON.OpenDataServer.1
ServerProjectName		Optional filename, which if specified causes the OPC Server to use the specified file, if supported by the server.

Script Interface

The Script Interface defines the methods for the OPC communications control.

Functions

Value	Function for getting and setting an OPC item value.
Read	Function to read the value of an OPC item.
Write	Function to write the value of an OPC item.

Value

Reads or writes the value of an OPC item.

Example 1 – Reading a value:

intVal = OMRONCXOPCCommunicationsControl1.Value("MyGroup", "BoilerTemp")

In this example, the OPC item 'BoilerTemp' in the OPC group called "MyGroup" will be read from the OPC Server and will be stored in 'intVal'.

Example 2 – Writing a value:

OMRONCXOPCCommunicationsControl1.Value("MyGroup", "BoilerTemp") = 50

In this example, the value 50 will be written to the OPC item 'BoilerTemp'.

Note: 'Value' is the default property so is assumed if omitted. Therefore, the following examples are the same:

intVal = OMRONCXOPCCommunicationsControl1.Value("MyGroup", "BoilerTemp")
and
intVal = OMRONCXOPCCommunicationsControl1 ("MyGroup", "BoilerTemp")

Read

Reads the value of an OPC item.

Example of synchronous read:

intVal = OMRONCXOPCCommunicationsControl1.Read("MyGroup", "BoilerTemp")

In this example, the OPC item 'BoilerTemp' in the OPC group called "MyGroup" will be read from the OPC Server and will be stored in 'intVal'. The script will wait for the read operation to complete before continuing to execute the next line. This is identical to the operation of the 'Value' method.

Write

Writes the value of an OPC item.

Example of synchronous write:

OMRONCXOPCCommunicationsControl1.Write "MyGroup", "BoilerTemp", NewValue

In this example, 'NewValue' will be written to the OPC item 'BoilerTemp' in the OPC group called "MyGroup". The script will wait for the write operation to complete before continuing to execute the next line. This is identical to the operation of the 'Value' method.

Appendix B
Lite Communications Control

This appendix contains a list of the available component properties and gives details of the Visual Basic script interface. These properties can be set in run time by using a Visual Basic script command – for example: -

```
OMRONCXCommunicationsControl1.ServerNodeName = "\\NAME"
```

The Script Interface defines the Visual Basic script interface for the Lite communications control. See ExecuteVBScript script functions for more information on running Visual Basic Script.

Component Properties

Property Title	Example	Description
DisplayErrors	True False	When set True, the object will display a message box for any errors. If set to False, error messages are not displayed.
ListPLCs		Holds a list of all PLC names configured in the project file. This property is read only.
ListPoints		Holds a list of all point names configured in the project file. This property is read only.

Script Interface

The Script Interface defines the Visual Basic script interface for the Lite communications control.

Functions

Value	Function for getting and setting an area of memory in a PLC. This function allows logical names to be used. If an array is used, the first element is returned.
Values	Function for getting and setting an area of memory in a PLC. This function allows logical names to be used. If an array is used then a SAFEARRAY is returned with all values.
SetDefaultPLC	Function for setting the default PLC. This is primarily used when a project contains multiple PLCs.
OpenPLC	Opens the specific PLC for communications.
ClosePLC	Closes the specific PLC.
ReadArea	Function for reading a block of memory from the PLC.
WriteArea	Function for writing a block of memory to the PLC.
RunMode	Function for reading / writing the current mode of the PLC.
TypeName	Function for reading the PLC type (e.g. CQM1H).
IsPointValid	Checks a point name is valid.
PLC Memory Functions	A, AR, C, CIO, D, DM, DR, E, EM, G, GR, H, IR, LR, SR, ST, T, TC, TK, W. Functions for getting and setting the memory areas in the PLC.

Value

Reads the value of an address from a PLC, or writes a value to an address in a PLC. This function allows logical names.

Example 1 – Reading a value from the PLC using a logical name.

intVal = OMRONCXCommunicationsControl1.Value("BoilerTemp")
or
intVal = OMRONCXCommunicationsControl1 ("BoilerTemp")

In these examples, the PLC address associated with 'BoilerTemp' will be read from the PLC and stored in 'intVal'. "Value" is the default property and does not have to be specified.

Example 2 – Writing a value to the PLC using a logical name.

OMRONCXCommunicationsControl1.Value("BoilerTemp") = 50
or
OMRONCXCommunicationsControl1 ("BoilerTemp") = 50

In these examples, the value 50 will be written to the PLC address associated with 'BoilerTemp'. "Value" is the default property and does not have to be specified.

Further examples

OMRONCXCommunicationsControl1 ("PLCName/DM0") = 1000
OMRONCXCommunicationsControl1 ("DM0") = 1000
intval = CommunicationsControl1("DM100")

These examples are physical addresses. A PLC name can optionally be specified otherwise the current default PLC will be assumed.

Values

Reads an array of values from a PLC, or writes an array of values to a PLC. This function allows logical names. If an array is used then a SAFEARRAY is returned with all values.

Example 1 – Reading an array of values from the PLC using a logical name.

SomeArray = OMRONCXCommunicationsControl1.Values("BoilerTemps")

Example 2 – Writing an array of values to the PLC using a logical name.

OMRONCXCommunicationsControl1.Values("BoilerTemps") = SomeArray

SetDefaultPLC

The 'SetDefaultPLC' function can be used to inform the script parser that a particular PLC is has been set as the default. Once a default PLC has been set, then it is not necessary (with some functions) to specify a PLC name. For example,

OMRONCXCommunicationsControl1.SetDefaultPLC("MyPLC")
intVal = OMRONCXCommunicationsControl1.Value("BoilerTemp1")

```
OMRONCXCommunicationsControl1.Value("BoilerTemp1") = 75
intVal = OMRONCXCommunicationsControl1.Value("DM50")
```

Each 'Value' function above will access data in the PLC called 'MyPLC'.

Note: If there is only 1 PLC in the project then it is not necessary to call the 'SetDefaultPLC' function. The first PLC in a project will automatically be set as the default PLC.

OpenPLC

Opens a PLC for communications. If no PLC is specified then the default PLC is opened.

Example 1:

```
OMRONCXCommunicationsControl1.SetDefaultPLC("MyPLC")
OMRONCXCommunicationsControl1.OpenPLC()
OMRONCXCommunicationsControl1.DM(100) = 10
OMRONCXCommunicationsControl1.DM(50) = 10
```

Example 2:

```
OMRONCXCommunicationsControl1.OpenPLC("MyPLC")
OMRONCXCommunicationsControl1.DM(100) = 10
```

ClosePLC

Closes a previously opened PLC. If no PLC is specified then the default PLC is closed.

Example:

```
OMRONCXCommunicationsControl1.ClosePLC("MyPLC")
```

ReadArea

Reads a specified block of memory from a PLC.

Examples of synchronous read:

```
MyVariant = OMRONCXCommunicationsControl1.ReadArea("MyPLC/DM0", 12, vbString)
MyVariant = OMRONCXCommunicationsControl1.ReadArea("BoilerTemp", 10, vbInteger)
MyVariant = OMRONCXCommunicationsControl1.ReadArea("BoilerTemp", 20)
```

In the first example, DM0 to DM11 will be read as characters (part of a string) from 'MyPLC' and will be stored in 'MyVariant'. The second example demonstrates that it is also possible to use a logical name for the start address, and that any VB variant types (such as vbInteger) can be used. The third example shows that the VB Variant type parameter is optional – if none is specified then vbInteger is assumed. The script will wait for the read operation to complete before continuing to execute the next line.

Note: If accessing from a non-Visual Basic language, like CX-Supervisor script, the following integral values should be used for the return type:

Constant	Value	Description

vbEmpty	0	Uninitialized (default)
vbNull	1	Contains no valid data
vbInteger	2	Integer subtype
vbLong	3	Long subtype
vbSingle	4	Single subtype
vbSingle	5	Double subtype
vbCurrency	6	Currency subtype
vbDate	7	Date subtype
vbString	8	String subtype
vbObject	9	Object
vbError	10	Error subtype
vbBoolean	11	Boolean subtype
vbVariant	12	Variant (used only for arrays of variants)
vbDataObject	13	Data access object
vbDecimal	14	Decimal subtype
vbByte	17	Byte subtype
vbArray	8192	Array

WriteArea

Writes a block of memory to a specified area in a PLC.

Examples of synchronous write:

```
MyString = "TestString"
OMRONCXCommunicationsControl1.WriteArea "MyPLC/DM50", 10, MyString
Dim newValue(2) As Long
newValue(1) = 0
newValue(2) = 1
OMRONCXCommunicationsControl1.WriteArea "BoilerTemp",2,newValue
```

In the first example, the contents of 'MyString' will be written into DM50 to DM54. Any additional data in 'MyString' will be ignored (i.e. if 'MyString' is 15 characters in length then the first 10 characters will be written to DM50 to DM54 and the remaining 5 characters will be ignored – {Note: each PLC address holds 2 characters}). The second example shows that a logical name can be used. The script will wait for the write operation to complete before continuing to execute the next line.

RunMode

Reads the current operating mode of a PLC (Stop/Program, Debug, Monitor, Run), where 0=Stop/Program mode, 1=Debug mode, 2=Monitor mode and 4=Run mode.

Example

intMode = OMRONCXCommunicationsControl1.RunMode("MyPLC")

In this example, the operating mode would be read from 'MyPLC' and stored in 'intMode'. If 'MyPLC' was in 'Monitor' mode then 'intMode' would be set to the value 2.

TypeName

Reads the PLC model name of a PLC (e.g. C200H, CQM1H, CVM1 etc).

Example

strPLCType = OMRONCXCommunicationsControl1.TypeName("MyPLC")

In this example, the PLC model type will be read from 'MyPLC' and will be stored in 'strPLCType'.

IsPointValid

Checks if a Point name has been defined in the CX-Server project file.

Examples

bValid = OMRONCXCommunicationsControl1.IsPointValid("MyPoint")

bValid = OMRONCXCommunicationsControl1.IsPointValid("MyPoint", "MyPLC")

In both examples, the boolean variable bValid is set True if the point "MyPoint" has been defined.

PLC Memory Functions (A, AR, C, CIO, D, DM, DR, E, EM, - G, GR, H, IR, LR, SR, ST, T, TC, TK, W)

All PLC memory functions (e.g. A, AR, D, DM etc.) work in exactly the same way. The following examples use the DM function to get and set the value of a DM address in a PLC.

Example 1

intVal = OMRONCXCommunicationsControl1.DM(100)

In this example, the contents of DM100 will be read from the PLC and stored in 'intVal'.

Note: These examples assume there is only 1 PLC in the CX-Server project file, or that the 'SetDefaultPLC' function has been used to select the required PLC. Refer to the 'SetDefaultPLC' function for details about using script with multiple PLCs in the project.

Example 2

OMRONCXCommunicationsControl1.DM(100) = 75

In this example, the value 75 will be written to DM100 in the PLC.

Bit addressing, that is accessing data from individual memory bits, is also supported by these memory areas: IR, AR, HR and CIO.

Example 3

bVal = OMRONCXCommunicationsControl1.IR("100.2")

In this example, the status of bit IR100.2 (i.e. bit 2 of IR100) will be read from the PLC and stored in 'bVal' (e.g. 'bVal' will be set to TRUE or FALSE).

Example 4

OMRONCXCommunicationsControl1.IR("100.2") = True

In this example, bit IR100.2 (i.e. bit 2 of IR100) in the PLC will be set to True. Note that use of the quotes is optional, but is required to differentiate between 100.1 and 100.10

APPENDIX C
JScript Features

This appendix provides a summary of JScript features available for use with the ExecuteJScript and ExecuteJScriptFile script functions. These features are provided by the Windows Scripting Host, included by default with Windows 98, Windows ME, Windows 2000 and Windows XP and installed by Internet Explorer 4.0 and later. For Windows 95 and Windows NT, the Windows Scripting Host is available as a free download from Microsoft's Web site.

For details of the latest versions and support contact Microsoft at http://msdn.microsoft.com/scripting

Category	Keyword / Feature
Array Handing	Array join, length, reverse, sort
Assignments	Assign (=) Compound Assign (OP=)
Booleans	Boolean
Comments	/*…*/ or //
Constants / Literals	NaN null true, false Infinity undefined
Control flow	break continue for for..in if…else return while
Dates and Time	Date getDate, getDay, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getYear, getUTCDate, getUTCDay, getUTCFullYear, getUTCHours, getUTCMilliseconds, getUTCMinutes, getUTCMonth, getUTCSeconds, setDate, setFullYear, setHours, setMilliseconds, setMinutes, setMonth, setSeconds, setTime, setYear, setUTCDate, setUTCFullYear, setUTCHours, setUTCmillisecinds, setUTCMinutes, setUTCMonth, setUTCSeconds, toGMTString, toLocaleString, toUTCString, parse, UTC
Declarations	function new this var with

Category	Keyword / Feature
Function Creation	Function arguments, length
Global Methods	Global escape, unescape eval isFinite, isNaN parseInt, parseFloat
Maths	Math abs, acos, asin, atan, atan2, ceil, cos, exp, floor, log, max, min, pow, random, round, sin, sqrt, tan, E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2
Numbers	Number MAX_VALUE, MIN_VALUE NaN NEGATIVE_INFINITY, POSITIVE_INFINITY
Object Creation	Object new constructor, prototype, toString, valueOf
Operators	Addition(+), Subtraction (-) Modulus arithmetic (%) Multiplication (*), Division (/) Negation (-) Equality (==), Inequality (!=) Less Than (<), Less Than or Equal To (<=) Greater Than (>) Greater Than or Equal To (>=) Logical And (&&), Or (||), Not (!) Bitwise And (&), Or (|), Not (~), Xor (^) Bitwise Left Shift (<<), Shift Right (>>) Unsigned Shift Right (>>>) Conditional (?:) Comma (,) delete, typeof, void Decrement (--), Increment (++)
Objects	Array Boolean Date Function Global Math Number Object String
Strings	String charAt, charCodeAt, fromCharCode indexOf, lastIndexOf split toLowerCase, toUpperCase length

APPENDIX D
Obsolete Features

This appendix provides a summary of features that are obsolete and have been removed from the standard documentation. Details are included here to assist maintaining old projects still using these features. These features should not be used in development of new solutions as it is likely support for the following features may and will be removed from the next or future releases.

Windows 95

This product is no longer tested under Windows 95, so Omron cannot guarantee correct operation. It is recommended to upgrade to Windows NT or NT successor.

DDE Commands

DDE as a means for exchanging data has now been obsolete for some years. In fact for so long even its successor, OLE Automation is obsolete. DDE has also proved to be a poor technology, suffering from unfixed memory leaks both in the native Operating Systems, and tools like Microsoft Excel. This technology has now been replaced and the CX-Supervisor Communications Control should be used instead.

The following DDE script commands are obsolete.

DDEExecute

Syntax

```
returnstate = DDEExecute(channel, {command})
```

Remarks

Argument	Type	Description
returnstate	Bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the channel based on the DDEInitiate() command must be open or an error is reported.
command	String	This is a *command* as recognised by the server application specified within the *channel*.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
DDEExecute(channelname, {[OPEN("C:\EXCEL\WORK\SHEET2.XLS")]})
```

The file 'SHEET2.XLS' within path 'C:\EXCEL\WORK' is opened in Microsoft Excel, as specified by the Integer point 'channelname'. The file 'SHEET1.XLS' is already open in Microsoft Excel

DDEInitiate

Syntax

```
channel = DDEInitiate("server", topic")
```

Remarks

Argument	Type	Description
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command.
server	String	This contains the application that supports DDE as a DDE server. Typically, this is the name of the applications' *.EXE executable file without the filename extension. At runtime, the *server* application must be open or a value cannot be returned and an error is reported.
topic	String	This contains the name of the topic recognised by the *server* application. Typically, a topic is a document within an application. At runtime, the *topic* must be open or a value cannot be returned and an error is reported. The *topic* may be left empty, which enables documents to open remotely prior to making a specified connection. The *topic* name 'System' may be used to find out which other topics within the *server* application are available. However, this is dependant on the *server* application supporting this *topic*.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
```

The Integer point 'channelname' is provided with a DDE link to the application Microsoft Excel which is run by the executable filename 'EXCEL.EXE', and to the file 'SHEET1.XLS' within that application.

DDEOpenLinks

Syntax

```
returnstate = DDEOpenLinks(channel)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the channel in the DDEInitiate() command must be open or an error is reported.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
DDEOpenLinks(channelname)
```

The DDEOpenLinks command enables points which have been configured to communicate via DDE to begin data transfer. Data transfer between CX-Supervisor and the application Microsoft Excel is automatically maintained until the channel is closed either by Microsoft Excel or by the command DDETerminate() using the Integer point 'channelname', or the command DDETerminateAll().

DDEPoke

Syntax

```
returnstate = DDEPoke(channel, "item", pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the in the DDEInitiate() command must be open or an error is reported.
item	string	This is an item as recognised by the server application. For instance, a cell is an *item* within a spreadsheet application. Likewise, a page is an *item* for a word processing application. It is wholly dependant on the server application
pointname	point	This is a point whose attributes must include a DDE Access of 'Read/Only' or 'Read/Write'. The contents of this point are assigned to the server application.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
DDEPoke(channelname, "R2C5", data)
```

The content of point 'data' is sent to row 2, column 5 of 'SHEET1.XLS' in the Microsoft application. The Microsoft Excel application, and 'SHEET1.XLS' are specified by Integer point 'channelname'.

DDERequest

Syntax

```
pointname = DDERequest(channel, "item")
```

Remarks

Argument	Type	Description
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the channel in the DDEInitiate() command must be open or an error is reported.
item	string	This is an item as recognised by the server application. For instance, a cell is an *item* within a spreadsheet application. Likewise, a page is an *item* for a word processing application. It is wholly dependent on the server application.
pointname	point	This is a point whose attributes must include a DDE Access of 'Read/Write'.

Typical Example

```
channelname = DDEInitiate("Excel", "Sheet1.xls")
cellref = DDERequest("channelname", "R2C5")
```

The point 'cellref' is filled from a specific item, row 2, column 5 from 'SHEET1.XLS' from the Microsoft Excel application, specified by the Integer point 'channelname'.

DDETerminate

Syntax

```
returnstate = DDETerminate(channel)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
channel	Integer point	This is an integer point which contains the return value of the DDEInitiate() command. Both server and topic parameters applied to the channel in the DDEInitiate() command must be open or an error is reported.

Typical Example

```
DDETerminate(channelname)
```

The server and topic specified by Integer point 'channelname' is closed.

DDETerminateAll

Syntax

```
returnstate = DDETerminateAll()
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.

Typical Example

```
DDETerminateAll()
```

All previously initiated DDE links are closed.

EnableDDE

Syntax

```
returnstate = EnableDDE(pointname)
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
Pointname	bool point	A Boolean point that holds the required enable/disable state

Typical Examples

```
EnableDDE(result)
```

DDE functions are enabled based on the value of point 'result'. If 'point' is 'TRUE', then DDE is enabled, if 'point' is 'FALSE', then DDE is disabled.

```
EnableDDE(TRUE)
```

DDE functions can also be enabled directly without using a point to hold the desired status.

EditGraph

Syntax

```
returnstate = EditGraph("graphid")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend graph to be edited.

Typical Example

```
EditGraph("Graph_1")
```

The Edit Graph dialog is displayed offering options to view historical data for the chosen trend graph.

- **Display Data** loads the currently selected data sample i.e. either the current screen data or a snapshot of the data, into the trend graph.
- **Snapshot** stores the current data buffer associated with the trend graph. The snapshot is given a time stamped default description.
- **Description** provides the ability to change the description associated with the snapshot.
- **Import Data** provides the ability to load in a previously saved trend graph file.
- **Export Data** provides the ability to store a snapshot to a file, either in internal CX-Supervisor format, or as a text file that can be imported into other applications.
- **Delete** removes the currently selected snapshot.

Note: This command is provided for compatibility with SCS v2.0 applications. For newer applications the data logging facilities should be used in preference.

Note: This command can only be used if the trend is set to log to a file.

SaveGraph

Syntax

```
returnstate = SaveGraph("graphid")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend to be saved.
pagename	string	*Optional* parameter indicating the name of the page that the graph is on.

Typical Examples

```
SaveGraph("Graph_1", "TestPage1")
```

The trend graph on the page 'TestPage' with the identifier 'Graph_1' has its data saved to disc.

```
SaveGraph("Graph_2")
```

The trend graph on the current page with the identifier 'Graph_2' has its data saved to disc.

Snapshot

Syntax

```
returnstate = Snapshot("graphid", "pagename")
```

Remarks

Argument	Type	Description
returnstate	bool	*Returnstate* is '1' if the function is successful, or '0' otherwise.
graphid	string	The identifier of the trend graph to have a snapshot
pagename	string	*Optional* parameter indicating the name of the page that the graph is on.

Typical Examples

```
Snapshot("Graph_1", "TestPage1")
```

The current data in trend graph 'Graph1' on 'TestPage1', is stored and is able to be viewed via the EditGraph command.

```
Snapshot("Graph_2")
```

The current data in trend graph 'Graph1' on the current page, is stored and is able to be viewed via the EditGraph command.

Note: This command is provided for compatibility with SCS v2.0 applications. For newer applications the data logging facilities should be used in preference.

GLOSSARY OF TERMS

ADO

ADO stands for Active Data Objects and is data access technology which uses OLE-DB to access data sources in a uniform way e.g. MS-Access databases, MS-Excel spreadsheets and Comma Separated Variable files.

AND

A logic operator used to interrogate Boolean type points. AND returns 'TRUE' if all arguments are 'TRUE'. An example of AND is that if *a* is a statement and *b* is a statement, AND returns 'TRUE' if both *a* and *b* are 'TRUE'. If one or both statements return 'FALSE' then AND returns 'FALSE'.

Application

A software program that accomplishes a specific task. Examples of applications are CX-Supervisor, SYSMAC-CDM, Microsoft Word for Windows and Microsoft Excel. CX-Supervisor and its development environment allows the creation and testing of new applications through a Graphical User Interface (GUI).

Arguments

Words, phrases, or numbers that can be entered on the same line as a command or statement to expand or modify the command or statement within the CX-Supervisor script language. The command acts on the argument. In essence the command is a *verb*, and the argument is the *object of the verb*. An example of an argument in CX-Supervisor is "`Message("Text")`" where `Message` is a command within the script language, and `"Text"` is the argument upon which the command will act.

ASCII

An old standard, defining a set of characters. Officially using only 7 bits allows definitions for only 127 characters, and does not include any accented characters.

Bitmap

The representation of an image stored in a computer's memory. Each picture element (pixel) is represented by bits stored in the memory. In CX-Supervisor a bitmap image can be installed as a single object.

Boolean type

A type of point where the value of the point can be one of two states. Essentially the two states are '0' and '1', but these states can be assigned a meaningful designation. Examples are:

State	Example	Example	Example	Example
0	'OFF'	'FALSE'	'OUT'	'CLOSED'
1	'ON'	'TRUE'	'IN'	'OPEN'

See also: **AND**, **NOT** and **OR**.

COM

COM is a Microsoft technology that allows components used to interact.

Communications Driver The relevant communications management system for OMRON PLCs in conjunction with Microsoft Windows, providing facilities for other SYSMAC software to maintain PLC device and address information and to communicate with OMRON PLCs and their supported network types.

Constant Within CX-Supervisor, a constant is a point within the script language that takes only one specific value.

Control Object In CX-Supervisor, a control object is applied in the development environment and can be a pushbutton, a toggle button, a slider, a trend graph, a rotational gauge or a linear gauge. Essentially a control object can be a complex graphic object consisting of a number of primitive graphic objects, which provides user interaction.

CX-Server An advanced communications management system for OMRON PLCs providing facilities for software to maintain PLC device and address information and to communicate with OMRON PLCs and their supported network types. CX-Server supports CS-Series PLCs.

Database connection A Database connection (or Connection for short) contains the details used to access a data source. This can either be via Data Source Name (DSN), filename or directory.

Database Connection Level A Database Connection Level is a string which determines what level in the database tree hierarchy is to be operated on. Some examples are listed below:

"Northwind"	Connectionlevel
"CSV.Result"	Recordset level
"Northwind.Order Details.OrderID"	Field level
"Invoice.Data Types"	Schema level

Database Recordset A Database recordset (or Recordset for short) is a set of records. This could either be an actual Table in the database, or a table that has been generated as a consequence of running a Query.

Database Schema A Database Schema (or Schema for short) obtains database schema information from a Provider.

Database Server Query A Database Server Query (or Server Query for short) is a query that is stored in the actual Database. They are pre-defined and added by the database designer which means they are 'fixed' for the duration of a project. Server Queries may have pre-defined 'Parameters', which allow criteria to be passed to the query at runtime e.g. values to filter, allowing one query to be used to produce different results. Each pre-defined parameter must have a Parameter Association defined. Because these queries are stored in a compiled and tested form they are more efficient and therefore preferential to running a SQL Query.

Database SQL Query A Database SQL Query (or SQL Query for short) is interpreted dynamically at runtime. The SQL Text can be modified at runtime, enabling different Queries to be run for varying situations however, the SQL Text has to be compiled on the fly every time it is executed and consequently is less efficient than a Server Query.

DBCS DBCS stands for Double Byte Character Set and is a Microsoft extension of ASCII which uses 2 bytes (16 bits) to define character codes. With this larger range it can include accented characters, extended ASCII characters, Nordic characters and symbols.

DCOM DCOM is a distributed version of COM that allows components on different PCs to interact over a network.

DDE Dynamic Data Exchange. Now obsolete. A channel through which correctly prepared programs can actively exchange data and control other applications within Microsoft Windows.

Development Environment SCADA applications are created and tested using the development environment within CX-Supervisor. On completion, the finished application can be delivered as a final customer application to be run by the run-time environment.

DLL Dynamic Link Library. A program file that although cannot be run stand-alone as an executable, can be utilised by one or more applications or programs as a common service. DLL files have a *.DLL extension. DLL's comprise a number of stand-alone functions. In CX-Supervisor, a DLL containing icons can be accessed to represent the display part of an OLE object. One such DLL, 'MORICONS.DLL', is provided in the standard Microsoft Windows installation.

Download A recipe is *downloaded* during runtime. This process involves identifying the appropriate recipe and executing the validation code, if any exists. The download is complete when each ingredient has set its point to the target value.

Executable A file that contains programs or commands of an application that can be executed by a user or another application. Executable files have a *.EXE file extension. CX-Supervisor provides two executable files, one for the development environment (CXSUPERVISORDEV.EXE), and one for the run-time environment (SCS.EXE).

Expressions In the CX-Supervisor script language, expressions are a construct for computing a value from one or more operands. For instance, in the example "`lift = height + rate`", the expression is "`height + rate`" where the result yielded from the expression is used for the value of "`lift`". Outside of the script language, expressions consisting of operators and operands can be used to control objects , through actions.

Field association	A field association enables a link to be made between a CX-Supervisor Point and a particular field (i.e. column) within a recordset.
Graphic Object	In CX-Supervisor, a graphic object is created in the development environment, and can be a line, an arc, a polygon (including a square and rectangle), a round rectangle, an ellipse (including a circle), or a polyline. A complex object can exist as a combination of two or more graphic objects.
GUI	Graphical User Interface. Part of a program that interacts with the user and takes full advantage of the graphics displays of computers. A GUI employs pull-down menus and dialog boxes for ease of use. Like all Microsoft Windows based applications, CX-Supervisor has a GUI.
I / O type	Input / Output type. An attribute of a point that defines the origin and destination of the data for that point. The data for a point can originate (be *input* from) and is destined (is *output* to) to the internal computer memory, or PLC.
Icon	Pictorial representations of computer resources and functions. The CX-Supervisor development environment and run-time environment are run from icons.
Ingredient	Each recipe consists of at least one ingredient. Each ingredient must be related to an existing point.
Integer type	A type of point where the value of the point can only be a whole positive or negative number.
Item	Within the CX-Supervisor script language, Item is a generic term for a point, OPC item or Temperature Controller item.
JScript	A Java style scripting language supported by Microsoft's Windows Scripting Host.
JVM	Java Virtual Machine.
Microsoft Excel	A spreadsheet application.
Microsoft Windows	A windowing environment that is noted for its GUI, and for features such as multiple typefaces, desk accessories (such as a clock, calculator, calendar and notepad), and the capability of moving text and graphics from one application to another via a clipboard. CX-Supervisor will run only under Microsoft Windows.
Microsoft Word for Windows	A word processing application.
Nesting	To incorporate one or more IF THEN ELSE/ELSEIF ENDIF statements inside a structure of the same kind.

Network

1. Part of the PLC configuration, based on the device type. The number of Networks available is dependant on the device type.
2. A number of computers linked together with a central processing point known as a Server which is accessible to all computers. Networks affect CX-Supervisor in that further Network associated options are available if the computer is Network connected.

Non-Volatile

A point that is designated as 'non-volatile' is a point whose value is saved on disk and automatically reloaded when CX-Supervisor resumes execution.

NOT

A logic operator used to interrogate Boolean type points which produces the Boolean inverse of the supplied argument. An example of NOT is that if *a* is a statement and is 'FALSE', then NOT returns 'TRUE'. If *a* is a statement and is 'TRUE', then NOT returns 'FALSE'.

Object

In CX-Supervisor, an object can be text, graphics, a control, a bitmap, or ActiveX object as created in the development environment. A complex object can exist as a combination of two or more objects of any of the above types. Specifically, graphical objects can be categorised as a line, an arc, a polygon (including a square and rectangle), a round rectangle, an ellipse (including a circle), or a polyline. A control is essentially a complex graphic object and is specifically either a pushbutton, a toggle button, a slider, a trend graph, a rotational gauge or a linear gauge.

OLE-DB

OLE-DB is the underlying database technology, on which ADO relies. OLE-BD is designed to be the successor to ODBC.

Operand

The term used for constants or point variables.

Operator

A symbol used as a function, with infix syntax if it has two arguments (e.g. "+") or prefix syntax if it has only one argument (e.g. NOT). The CX-Supervisor script language uses operators for built-in functions such as arithmetic and logic.

OR

A logic operator used to interrogate Boolean type points. OR returns 'TRUE' if any of the supplied arguments are 'TRUE'. An example of OR is that if *a* is a statement and *b* is a statement, OR will return 'TRUE' if either *a* and *b* are 'TRUE'. If both statements return 'FALSE' then OR will return 'FALSE'.

Pages

The combination and manipulation of pages containing objects within projects forms the basis of CX-Supervisor. More than one page can exist for each project. The pages in a project provide the visual aspect of CX-Supervisor corresponding to a display with the objects contained in each page providing a graphical representation of the system being monitored.

Parameter Association A Parameter Association enables values, either constant or stored in a point, to be passed to a Server Query.

Pixel A single displayable point on the screen from which a displayed image is constructed. The screen resolution of the computer's Visual Display Unit (VDU) is defined by the number of pixels across and the number of pixels down (e.g. 1024 x 768).

See also SVGA mode and VGA mode.

PLC Programmable Logic Controller.

Point variable A point within the CX-Supervisor script language that stores a value or string assigned to that point.

Point A point is used to hold a value of a predefined type - Boolean, Integer, Text, etc. The contents of a point may be controlled by a graphical object or I/O mechanism such as PLC communication. The contents of a point may control the action or appearance of an object, or be used for output via an I/O mechanism.

Program Manager An integral part of Microsoft Windows 3.x which allows Microsoft Windows based applications to be started from icons and for all applications to be organised. CX-Supervisor can be run from Program Manager.

Project A CX-Supervisor application will consist of one or a number of pages linked together. The pages may contain passive or active graphics, text or animations, and may be grouped together logically to form a project. A project may consist of many pages, or simply a single page. Projects may be built and tested within the CX-Supervisor development environment, and run stand-alone under the CX-Supervisor run-time environment.

Only one project at a time may be open for editing within the CX-Supervisor development environment.

Real type A type of point where the value of the point can be any number, including those containing a decimal point.

Recipe A recipe is a set of pre-defined steps used to perform a particular task. A CX-Supervisor project may contain zero or more number of recipes. Recipes are defined in the development environment and executed, or downloaded, in the run-time environment.

Run Time Environment SCADA applications are run using the run-time environment of CX-Supervisor, following creation of the application in the CX-Supervisor development environment.

SCADA Supervisory Control and Data Acquisition. (see CX-Supervisor)

Server A Server is the central processing point of a Network which is accessible to all computers. Networks affect CX-Supervisor in that further associated options are available if the computer Network is connected.

Server Application An application that can be used to view or interact with, whilst currently within CX-Supervisor.

Statement Within the CX-Supervisor script language, a statement is a command understood by the run-time environment. Statements are constructed of commands and arguments, which when combined, help to formulate a finished application to be used in the run-time environment.

String The contents of a Text type point that can only contain literal alphanumeric characters. A string starts following an opening quotation mark, and ends before a closing question mark; in the example "`name = "spot"`", the point "`name`" holds the string `spot`.

SVGA mode A mode of video display that provides 800 × 600 pixel resolution (or higher) with 16 or more colours and is supported on Super Video Graphics Adapter systems.

CX-Supervisor A SCADA software application which creates and maintains graphical user interfaces and communicates with PLCs and other I/O mechanisms.

Target Value An ingredient must specify a target value for its related point. This is the value to which the point will be set in runtime when the recipe is downloaded.

Taskbar An integral part of Microsoft Windows which allows Microsoft Windows based applications to be started. CX-Supervisor is run from the Taskbar.

Text Object In CX-Supervisor, a text object is a string on a page. Attributes such as typeface, point size, embolden, italicise, underline, left justify, flush right, and centre can be applied to enhance its presentation.

Text type A type of point that holds a string.

Unicode A Multi-Byte Character Set, which not only includes European Characters like DBCS, but can also include global support including for Japanese, Chinese and Cyrillic fonts. However, Unicode is not supported on all Windows platforms.

Validation Code Recipe validation code is CX-Supervisor script language which is used to check point values before downloading a recipe.

VGA mode A mode of video display that provides 640 × 480 pixel resolution with 16 colours and is supported on Video Graphics Adapter systems.

VBScript A Visual Basic style scripting language supported by Microsoft's Windows Scripting Host.

VGA mode	A mode of video display that provides 640 × 480 pixel resolution with 16 colours and is supported on Video Graphics Adapter systems.
Windows Desktop	An integral part of Microsoft Windows which allows Microsoft Windows based applications to be started from icons and for all applications to be organised. CX-Supervisor can be run from Windows Desktop.
Windows Scripting Host	A scripting engine supplied by Microsoft to run VBScript or JScript. See http://msdn.microsoft.com/scripting
Wizard	Wizards are dialogs used by the CX-Supervisor development environment to take the user through complex operations in a simplified step-by-step process.

INDEX

A

AcknowledgeAlarms - *Alarm Commands*: · 78
AcknowledgeAllAlarms - *Alarm Commands*: · 78
AcknowledgeLatestAlarm - *Alarm Commands*: · 79
Alarm Commands · 78
    AcknowledgeAlarms · 78
    AcknowledgeAllAlarms · 78
    AcknowledgeLatestAlarm · 79
    ClearAlarmHistory · 79
    CloseAlarmHistory · 79
    CloseAlarmStatus · 80
    DisplayAlarmHistory · 80
    DisplayAlarmStatus · 81
    EnableAlarms · 81
    Is AlarmAcknowledged · 82
    IsAlarmActive · 82
Alarms
    Script Editor · 78–83
Alias Examples · 29
Animation Editor
    Expressions · 3, 161
Animations
    Expressions within · 3
Appendix A
    OPC Communications Control · 144
Appendix B
    Lite Communications Control · 146
Appendix C
    JScript Features · 152
Appendix D
    Obsolete Features · 154
Application · 159
Arguments · 159
Arithmetic Operators - *Logic and Arithmetic*: · 12

B

Balloon Script - *Script Examples*: · 138
Basic Point Assignment – *Points:* · 10
BCD - *Text Commands*: · 98
Bin - *Text Commands*: · 98
Bitmap · 159
    Pixel · 164
Bitwise Operators - *Logic and Arithmetic*: · 12
Blink - *Object Commands*: · 42

C

Call - *Subroutines*: · 22
CancelForce - *Point Commands*: · 58
Case Selected - *Control Statements*: · 19
Chr - *Text Commands*: · 98
ClearAlarmHistory - *Alarm Commands*: · 79
ClearErrorLog - *Event/Error Commands*: · 104
ClearGraph -*Graph Commands*: · 90
ClearLogFile - *Data Logging Commands*: · 112
ClearSpoolQueue - *Printer Commands*: · 106
Close Page - *Page Commands*: · 50
CloseAlarmHistory - *Alarm Commands*: · 79
CloseAlarmStatus - *Alarm Commands*: · 80
CloseComponent - *Communications Commands*: · 56
CloseErrorLog - *Event/Error Commands*: · 104
CloseFile - *File Commands*: · 83
CloseLogFile - *Data Logging Commands*: · 112
CloseLogView - *Data Logging Commands*: · 113
Colour - *Object Commands*: · 42
Colour Palette · 142
COM · 159
Command String Delimiters - *Punctuation*: · 23
Communications Commands · 56
    CloseComponent · 56
    EnableOLE · 56
    EnablePLC · 57
    OpenComponent · 57
Communications Drive · 160
Constant · 160
Control Object · 160
Control Statements · 15
    Case Selected · 19
    Do While/Until Loop · 21
    For...Next Loop · 21
    Nested Conditional Statements · 17
    Simple Conditional Statements · 15
Conventions in this manual · 1

CopyArray - *Point Commands*: · 58
CopyFile - *File Commands*: · 83
Current Object - *Object Commands*: · 39
CX-Server · 160
CX-Supervisor · 165

D

Data Logging Commands · 112
ClearLogFile · 112
CloseLogFile · 112
CloseLogView · 113
ExportAndViewLog · 113
ExportLog · 114
OpenLogFile · 116
OpenLogView · 116
StartLogging · 117
StopLogging · 117
DBCS · 161
DCOM · 161
DDE · 161
DeleteFile - *File Commands*: · 84
Development environment · 161
Disable - *Object Commands*: · 44
DisableGroup - *Point Commands*: · 59
DisablePoint - *Point Commands*: · 59, 60
Display Page - *Page Commands*: · 49
DisplayAlarmHistory - *Alarm Commands*: · 80
DisplayAlarmStatus - *Alarm Commands*: · 81
DisplayErrorLog - *Event/Error Commands*: · 105
DisplayRecipes - *Recipe Commands*: · 94
DLL · 161
Do While/Until Loop - *Control Statements*: · 21
Double Byte Character Set · *See* DBCS
Download · 161
DownloadPLCProgram - *PLC Commands*: · 68
DownloadRecipe - *Recipe Commands*: · 94, 95
Dynamic Data Exchange · *See* DDE
Dynamic Link Library · *See* DLL

E

EditFile - *File Commands*: · 84
EditGraph - *Graph Commands*: · 91
EnableAlarms - *Alarm Commands*: · 81
EnableErrorLogging - *Event/Error Commands*: · 105
EnableGroup - *Point Commands*: · 61
EnableOLE - *Communications Commands*: · 56
EnablePLC - *Communications Commands*: · 57
EnablePoint - *Point Commands*: · 61
EnablePrinting - *Printer Commands*: · 107
Event/Error Commands · 104
ClearErrorLog · 104
CloseErrorLog · 104
DisplayErrorLog · 105
EnableErrorLogging · 105
LogError · 105
LogEvent · 106
Exponential - *General Commands*: · 50, 54
ExportAndViewLog - *Data Logging Commands*: · 113
ExportLog - *Data Logging Commands*: · 114
Expressions · 3

F

File Commands · 83
CloseFile · 83
CopyFile · 83
DeleteFile · 84
EditFile · 84
FileExists · 85
MoveFile · 85
OpenFile · 86
PrintFile · 86
Read · 87
ReadMessage · 87
SelectFile · 88
Write · 89
WriteMessage · 90
FileExists - *File Commands*: · 85
For...Next Loop - *Control Statements*: · 21
Force - *Point Commands*: · 61
ForceReset - *Point Commands*: · 62
ForceSet - *Point Commands*: · 62
FormatText - *Text Commands*: · 99
Functions and Methods · 9, 35
Further Point Assignment – *Points:* · 11

G

General Commands · 50
Exponential · 50, 54
PlayOLE · 51
PlaySound · 52
Rand · 52
RunApplication · 53
RunHelp · 53
ShutDown · 54
Sleep · 55
GenerateReport - *Report Commands*: · 95
GetBit - *Point Commands*: · 63
GetPLCMode - *PLC Commands*: · 68
GetPointValue - *Point Commands*: · 63
GetSpoolCount - *Printer Commands*: · 107
GetTextLength - *Text Commands*: · 100
Glossary of Terms · 159
Graph Commands · 90
ClearGraph · 90
EditGraph · 91
Snapshot · 92
StartGraph · 93
StopGraph · 92, 93
Graphic Object · 162
Graphical User Interface · *See* GUI
GUI · 162

H

Height - *Object Commands*: · 44
Hex - *Text Commands*: · 100
Horizontal Fill - *Object Commands*: · 45

I

Icons · 162
Indentation - *Punctuation*: · 24
Indirection within Script Commands and Expressions · 26
Ingredient · 162
InitiateArray - *Point Commands*: · 64
Input Point - *Point Commands*: · 64
Input/Output type · 162
Is AlarmAcknowledged - *Alarm Commands*: · 82
IsAlarmActive - *Alarm Commands*: · 82
Item · 162

J

Java Script · *See* JScript
Java Script Features · 152
JScript · 162
JScript Features · 152

L

Left - *Text Commands*: · 101
Lite Communications Control · 146
LogError - *Event/Error Commands*: · 105
LogEvent - *Event/Error Commands*: · 106
Logic and Arithmetic · 12
Arithmetic Operators · 12
Bitwise Operators · 12
Relational Operators · 14
Logical Operators - *Logic and Arithmetic*: · 13
Login - *Security Commands*: · 110
Logout - *Security Commands*: · 111

M

Message - *Text Commands*: · 101
Microsoft
Excel · 162
Windows · 162
Word for Windows · 162
Mid - *Text Commands*: · 102
Miscellaneous Commands
Remarks · 25
Move - *Object Commands*: · 46
MoveFile - *File Commands*: · 85
Multiple Commands - *Punctuation*: · 24

N

Nested Conditional Statements - *Control Statements*: · 17
Nesting · 162
Non-volatile · 163

O

Object · 163
Object Commands · 39
  Blink · 42
  Colour · 42
  Current Object · 39
  Disable · 44
  Height · 44
  Horizontal Fill · 45
  Move · 46
  Other Objects · 40
  Rotate · 46
  Vertical Fill · 47
  Visible · 48
  Width · 49
Objects – *Scripts:* · 7
Obsolete Features · 154
OPC Communications Control · 144
OpenComponent - *Communications Commands*: · 57
OpenFile - *File Commands*: · 86
OpenLogFile - *Data Logging Commands*: · 116
OpenLogView - *Data Logging Commands*: · 116
Other Objects - *Object Commands*: · 40
OutputPoint - *Point Commands*: · 65

P

Page – *Scripts:* · 7
Page Commands · 49
  Close Page · 50
  Display Page · 49
Pages · 163
Parenthesis - *Punctuation*: · 25
Pixel · 164
PlayOLE - *General Commands*: · 51
PlaySound - *General Commands*: · 52
PLC · 164
  Network · 163
PLC Commands · 67
  DownloadPLCProgram · 68
  GetPLCMode · 68
  PLCCommsFailed · 70
  PLCMonitor · 70
  SetPLCMode · 70
  SetPLCPhoneNumber · 71
  UploadPLCProgram · 72
PLC Memory Functions · 150
PLCCommsFailed - *PLC Commands*: · 70
PLCMonitor - *PLC Commands*: · 70
Point · 164
Point Arrays within Script Commands and Expressions · 27
Point Commands · 58
  CancelForce · 58
  CopyArray · 58
  DisableGroup · 59
  DisablePoint · 59, 60
  EnableGroup · 61
  EnablePoint · 61
  Force · 61
  ForceReset · 62
  ForceSet · 62
  GetBit · 63
  GetPointValue · 63
  InitiateArray · 64
  Input Point · 64
  OutputPoint · 65
  PointExists · 65
  SetBit · 66
  SetPointValue · 66
Point Variable · 164
PointExists - *Point Commands*: · 65
Points · 10
  Basic Point Assignment · 10
  Boolean · 159
  Further Point Assignment · 11
  Integer · 162
  Real · 164
  Script Editor · 10–11
  Text · 165
PrintActivePage - *Printer Commands*: · 108
Printer Commands · 106
  ClearSpoolQueue · 106
  EnablePrinting · 107

GetSpoolCount · 107
PrintActivePage · 108
PrintPage · 108
PrintScreen · 108
PrintSpoolQueue · 109
SetPrinterConfig · 109
PrintFile - *File Commands*: · 86
PrintMessage - *Text Commands*: · 102
PrintPage - *Printer Commands*: · 108
PrintReport - *Report Commands*: · 96
PrintScreen - *Printer Commands*: · 108
PrintSpoolQueue - *Printer Commands*: · 109
Program Manager · 164
Programmable Logic Controller · *See* PLC
Project · 164
Project – *Scripts:* · 7
Punctuation
Command String Delimiters · 23
Indentation · 24
Multiple Commands · 24
Parenthesis · 25
Quotation Marks · 25

Q

Quotation Marks - *Punctuation*: · 25

R

Rand - *General Commands*: · 52
Read - *File Commands*: · 87
ReadMessage - *File Commands*: · 87
Recipe · 164
Recipe Commands · 94
DisplayRecipes · 94
DownloadRecipe · 94, 95
Recipes
Download recipe · 165
Relational Operators - *Logic and Arithmetic*: · 14
Remarks - *Miscellaneous Commands*: · 25
Report Commands · 95
GenerateReport · 95
PrintReport · 96
ViewReport · 97
Return - *Subroutines*: · 23
Right - *Text Commands*: · 102
Rotate - *Object Commands*: · 46
RunApplication - *General Commands*: · 53
RunHelp - *General Commands*: · 53
Runtime Environment · 164

S

SCADA · 164
Script Editor
AND statement · 159
Applications, use of external · 1
Arithmetic functions · 12–15
Conditional statements, nesting · 17–19
Control Statements · 15–22. *See also* Control Statements
Current object statement · 39
Examples · 138
Executable files, use of · 1, 155, 161
FALSE Boolean state · 1, 4
Logical functions · 13–14
Mathematical precedence · 11, 12
Multiple statements on one line · 24
Nesting conditional statements · 17–19
NOT statement · 163
Object Commands · 39. *See also* Object Commands
Operator and operand · 1, 163
OR statement · 163
Parenthesis · 12, 25
Quotation marks · 24
Relational functions · 14–15
Script code examples · 138
Subroutines · 22–23. *See also* Subroutines
TRUE Boolean state · 1, 4
Script Examples · 138
Balloon Script · 138
Script Interface · 144, 146
Functions · 144, 146
PLC Memory Functions · 150
Script Interface Functions
ClosePLC · 148
OpenPLC · 148
Read · 145
ReadArea · 148
RunMode · 149
SetDefaultPLC · 147

TypeName · 150
Value · 144, 147
Values · 147
Write · 145
WriteArea · 149
Scripts · 7
Objects · 7
Page · 7
Project · 7
Security Commands · 110
Login · 110
Logout · 111
SetupUsers · 111
SelectFile - *File Commands*: · 88
Server · 165
Server Application · 165
SetBit - *Point Commands*: · 66
SetPLCMode - *PLC Commands*: · 70
SetPLCPhoneNumber - *PLC Commands*: · 71
SetPointValue - *Point Commands*: · 66
SetPrinterConfig - *Printer Commands*: · 109
SetupUsers - *Security Commands*: · 111
ShutDown - *General Commands*: · 54
Simple Conditional Statements - *Control Statements*: · 15
Sleep - *General Commands*: · 55
Snapshot - *Graph Commands*: · 92
StartGraph - *Graph Commands*: · 93
StartLogging - *Data Logging Commands*: · 117
Statement · 165
StopGraph - *Graph Commands*: · 92, 93
StopLogging - *Data Logging Commands*: · 117
String · 165
Subroutines · 22
Call · 22
Return · 23
Super Video Graphics Adapter · *See* SVGA
Supervisory Control and Data Acquisition · *See* SCADA
SVGA · 165

T

Target Value · 165
TCAutoTune - *Temperature Controller Commands*: · 72
TCBackupMode - *Temperature Controller Commands*: · 73
TCGetStatusParameter - *Temperature Controller Commands*: · 73
TCRemoteLocal - *Temperature Controller Commands*: · 74
TCRequestStatus - *Temperature Controller Commands*: · 75
TCReset - *Temperature Controller Commands*: · 77
TCRspLsp - *Temperature Controller Commands*: · 75
TCRunStop - *Temperature Controller Commands*: · 76
TCSaveData - *Temperature Controller Commands*: · 76
TCSettingLevel1 - *Temperature Controller Commands*: · 77
Temperature Controller Commands · 72
TCAutoTune · 72
TCBackupMode · 73
TCGetStatusParameter · 73
TCRemoteLocal · 74
TCRequestStatus · 75
TCReset · 77
TCRspLsp · 75
TCRunStop · 76
TCSaveData · 76
TCSettingLevel1 · 77
Text Commands · 98
BCD · 98
Bin · 98
Chr · 98
FormatText · 99
GetTextLength · 100
Hex · 100
Left · 101
Message · 101
Mid · 102
PrintMessage · 102
Right · 102
TextToValue · 103
ValueToText · 103
Text Object · 165
TextToValue - *Text Commands*: · 103
Typographical conventions · 1

U

Unicode · 165
UploadPLCProgram - *PLC Commands*: · 72
Using Aliases · 28

V

Validation Code · 165
ValueToText - *Text Commands*: · 103
VBScript · 31, 165
Vertical Fill - *Object Commands*: · 47
VGA · 165, 166
Video Graphics Adapter · 165. *See* VGA
ViewReport - *Report Commands*: · 97
Visible - *Object Commands*: · 48
Visual Basic · *See* VBScript
VJM · 162

W

Width - *Object Commands*: · 49
Windows Desktop · 166
Windows Scripting Host · 166
    JScript · 152
    VBScript · 31
Windows Taskbar · 165
Wizard · 166
Write - *File Commands*: · 89
WriteMessage - *File Commands*: · 90
WSH · *See* Windows Scripting Host